Date: June 19, 1989

To: Apple II Developers

From: The Apple II System Software Development Team

Subject: What's new between System Disk 4.0 and 5.0.

System Disk 5.0 Release Notes

OS Parts

ProBoot

Added 7 bytes onto the end of the status list since a 3rd party device was returning more bytes than we expected and was writing over the data following the list.

GetBootName routine modified to handle volume names with lower case letters.

GLoader

The OS_Kind flag is now set before setup files are executed instead of after.

The user ID is now set explicitly when shutting down temporary setup files in case the temporary setup file loaded a permanent setup file.

NDAs will now continue to be installed after the maximum number of CDAs have been installed.

GLoader now loads and executes the RESOURCE.MGR setup file immediately after the TOOL.SETUP file.

Added the ability to load a boot driver before starting up the device dispatcher.

Added a second entry point so that AppleShare (and other file systems) can prevent GLoader from performing certain startup procedures.

Added the ability for permanent setup files to request to be shutdown after they have been executed.

GLoader now loads and starts up the GS.OS.DEV file before initializing GS/OS.

GLoader now loads the EXPRESSLOAD file if it exists. The file is not loaded if we are running on a 512K system.

Modified the color palette used by the splash screen code in order to get rid of the green flash that was occurring just before the Finder desktop appeared. When the thermometer is drawn, the background is now the same as the Finder background instead of matching the border color.

When GLoader loads and executes a setup file, it will now allocate a 4K zero page/stack segment for the setup file to use if the file did not request its own zero page/stack segment from the system loader. The segment is deallocated after the setup file has executed.

GLoader now allocates $01/A000 - $01/BBFF for the System Loader and $01/BC00 - $01/BFFF for the Scatter Read code. The segments are allocated with a $32xx segment ID.

Before launching the start program, GLoader now resizes the segment allocated for the ERROR.MSG file in order to get rid of temporary messages which are only needed during initial boot.

If an FST returns a non-fatal error from its initialization routine, the FST will be unloaded and execution will continue. GLoader will not report the error since it is now the FST's responsibility to handle non-fatal errors. If the FST returns a fatal error, GLoader will report the error as before.

GLoader now does a case-insensitive compare when searching for a .SYS16 or .SYSTEM file.

GQuit

All OPEN calls are now done with "read" access instead of "as allowed" access in order to support AppleShare.

GQuit now calls the OS_event system service routine just before switching to ProDOS 8 and just after restarting GS/OS.

Prefix 33 (represented by "@") is now set whenever a GS/OS application is launched. If the application is being launched from an AppleShare volume, prefix 33 is set to the pathname provided by the AppleShare FST. Otherwise, prefix 33 is set to the same path that prefix 9 is set to.

When GQuit calls the System Loader to load a GS/OS application, it will now request that special memory not be used. If the application can't be loaded, GQuit will then make the call again, this time specifying that special memory can be used.

The system busy flag is now incremented on entry from a ProDOS 8 QUIT call and is decremented just before launching a ProDOS 8 or GS/OS application.

GQuit now handles a new bit in the Quit Flags. If bit 13 (the skip_std bit) is clear, prefixes 10, 11 and 12 are set to '.CONSOLE' when launching a GS/OS application. If the bit is set, then prefixes 10, 11 and 12 are left unchanged.

GQuit now ensures that the .CONSOLE device exists before launching a GS/OS application. If the device does not exist, a fatal error is reported.

GQuit now supports reloading GS/OS from memory. When quitting from a GS/OS application to a ProDOS 8 application, GQuit now informs the rest of GS/OS that a "warm" shutdown should be done. GQuit then attempts to copy the pieces of GS/OS which reside in banks $00 and $01 into non-special memory before launching ProDOS 8. When quitting from a ProDOS 8 application to a GS/OS application, GQuit copies the GS/OS pieces back into banks $00 and $01 and then informs the rest of GS/OS that a "warm" startup should be done. If GQuit is unable to obtain enough memory to store GS/OS in when switching to ProDOS 8, then it will force a system restart when switching from ProDOS 8 to GS/OS.

GQuit now makes a copy of the P8 file after loading it from disk. The copy is in non-special memory and is purgable. On subsequent switches to ProDOS 8, the file is loaded from memory instead of from disk unless it has been purged.

GQuit now checks a GS/OS application's auxtype to determine if it's "GS/OS aware" before attempting to launch it. If the application is not "GS/OS aware" and the path to the directory containing the app is > 64 characters, then GQuit displays a warning message since the app may not be able to run correctly.

GQuit now notifies the cache manager before launching a GS/OS application so that the cache will be purged if the system runs out of memory.

GQuit now checks the status of slot 3 before making the InitTextDev tool call and only makes the call if slot 3 is set internal since the Text Tool Set assumes that slot 3 is set internal.

GQuit now calls the Slot Arbiter when switching to/from ProDOS 8 in order to save/restore the the slot 3 screen holes.

The state of text page 2 shadowing is now preserved instead of being enabled.

When launching a GS/OS application, the prefixes are now set before the InitialLoad call is made so that the load file can reference other load files using prefix 1 (or 9).

When launching a GS/OS application, the cache_in_queue routine is now called before the application is loaded instead of after so that out-of-memory errors will be handled correctly.

System Loader Version 3.0

Skip Segments are only skipped by InitialLoad but not by a specific Load Segment function.

The ENTRY field in the Segment Header is now supported. This value is added to the starting location of the Segment to get the starting address during InitialLoad.

Segment Headers of any size are now supported.

OMF 2.1 is supported including Bank Relative and Skip segments.

Load Segment by Name did not support variable sized Segment Headers.

Starting Address added to Pathname Table so that Restart can support the ENTRY facility.

Relocation of non-SUPER records speeded up by checking for special cases.

When a Segment was unloaded, the purge level used was garbage (usually 1) instead of 3.

The Loader will load a Direct Page/Stack Segment even if No Special Memory specified by the caller to Initial Load.

At Initialization, the Loader will dispose of the memory block starting at $1A000 if it is assigned to the Loader. GLoad assigns memory location $1A000-1BBFF to the Loader for future use. Since the Loader currently does not need this memory it releases it.

After a Memory Load, the Loader was not setting the File Buffer size back to $4000. So the buffer size for subsequent Initial Loads was the size of the last Memory Loaded file.

Load Files with Application UserID's are no longer closed after an Initial Load if any Load Segments were skipped. This allows the Loader to load dynamic Load Segments and Run Time Library Load Segments without having to open and close the file for each Segment. Non Application Load Files (i.e. Desk Accessories, Init Files, etc.) are not affected. Application Load Files that have no Dynamic or Skip Load Segments are also not affected.

The InitialLoad function hung if the file requested was not there.

The LoaderInit call no longer zeroes out the Loader Globals. This allows GS/OS to restart the Loader without it initializing itself.

Unloading of a segment that was not loaded, caused system crash.

Made loadseg release temp data when loading from memory.

ExpressLoad

New for System Disk 5.0.

Fixed the infamous "DELETE bug" which caused a file to be left in a "half-deleted" state if an I/O error occurred at the wrong time during the destroy sequence. The fix causes the file's directory entry to be released if any deallocation has taken place, regardless of whether errors occurred during the process.

Fixed a bug where the ONLINE call did not mask off the unused high nibble of the unit_num parameter when deciding how many bytes to zero out in the user's data buffer. If the user called ONLINE with a unit_num of, for example, $01, P8 would only clear out 16 bytes (instead of 256 bytes) in the user buffer before proceeding correctly. The fix masks the unit_num byte properly and stores the resulting value for future use by the routine.

P8 now sets the video firmware "mode" byte ($4FB) to 0 and calls SETVID $FE93 again if running on a GS. This guarantees that the blinking "checkerboard" cursor will be set up correctly when BASIC (or any other SYSTEM program) gets control.

Expanded the size of the loader portion in order to make room for a GS QuickDraw patch. The loader portion of the code is of predefined size and was previously $B00 bytes long. This fact is hard-coded in this file, in the "ram.n file", and in the makefile, so expansion of this area is a non-trivial task that required some changes to be made by hand. Those changes are logged in a document titled "Loader Code Expansion Notes."

The patch, at label "InstallOSPatch," fixes a bug in the 5.0 System Disk where QuickDraw makes GS/OS™ calls without first checking that the System is installed in the machine. The patch installs code which is vectored into the GS/OS™ entry points and returns an error for GS/OS™ calls. This fixes things because QuickDraw handles GS/OS™ errors properly.

The patch is called just before the label "itsp8" and should only get executed when running on a GS and when P8 is being booted directly, i.e., not being installed by GQuit. The patch disables interrupts (for no specific reason, only as a precaution), switches the machine to native mode, allocates memory for the "real" GS/OS patch, copies that code into the allocated memory, sets up both GS/OS™ entry points to point to their appropriate pieces of the code, restores the environment and then returns.

Since these changes must be reflected in GQuit, the value of the GS/OS™ compatibility byte on $BF00 page has been changed from $01 to $02. Also, GQuit now checks the size of the P8 object file to ensure that it can find this byte in the first place. The size of the P8 file is now $3D7D bytes. The location of the GS/OS compatibility byte in the P8 object file has moved from $50FA to $51FA.

ProDOS FST

Changed the Create Call. The create call will now allow you to add a resource fork to an existing file. The only types of files that this is valid for is seedling sapling or tree. Furthermore, only the storage type and resource_eof fields are used by the create call when an extension request has been made. To extend a file, the application has to set the storage type to $8005 (Resource + $8000(extend bit)). This new feature brings along with it two new error codes. The error codes are $70 (resource_exist: the file already contains a resource fork) and $71 (res_add_err: The file cannot have a resource fork added, e.g. a subdirectory).

Fixed a problem with the Read Call. The read call would only issue multiblock read request if the blocks that were sequential were less the $8000.

Fixed a bug in GetDirEntry. If an active directory entry was found and the length of the name was zero(a damaged directory only) the FST was transferring 64K bytes to the user's buffer.

Changed the open routine. Now if the file is open for reading and another open call is made with "open_as_permitted" as the requested access I will open the file a second time giving read access to the caller.The last version would return an error.

The FST will now only display the "Volume Damaged" message once.

Fixed a bug in SetEOF. If SetEOF changed the structure of the file, it did not update the directory entry until the file was closed. This meant if you opened the file, did a set EOF followed by a delete call, the disk would be damaged.

Added option list support on all calls that have the parameter. The first data word of the option list contains the FST ID.

ProDOS FST now saves the case of file names.
Added two new FST specific commands. The new commands are:
(1) Set_Upper_Case
(2) Get_Upper_Case

The mix case bits are stored in the version and min version fields in the directory entry.

Fixed a major bug in the write routine. It was possible to have a block of data repeated into the next data block by mistake. I now copy the user's data into the I/O buffer even on multi-block transfer.

Fixed a bug in the ProDOS boot code. The boot code was reading block zero into the buffer when a sparse block was encountered.

Added Mod date to the volume. The four bytes preceding case bits in the volume header now contain the last mod date for the volume. Note: the following is how the Mod date is calculated for a ProDOS volume:

```
(1)     Mod_date := Create_date_From_Vol_Header
(2)     IF mod date <> NILL THEN
        Mod_Date := Mod_Date_From_Vol_Header
(3)     WHILE NOT End_of_Vol_Dir DO
                BEGIN
                Temp_Mod := Next_Entries_Mod_Date;
                If Temp_Mod > Mod_Date THEN
                        Mod_Dat := Temp_Mode
                END
```

The directory scanning is used in case the disk was modified by SOS,ProDOS 8, ProDOS 16 or GS/OS version 4.0, all of which, do not support the Modification Date in the volume header.

Caching

Added auto_flush_handler: If the cache manager cannot allocate memory for a block due to a cache full condition then the cache manager will automatically do a stop session followed by a start session if sessions were enabled.

The Cache Manager now adds itself to the OutOfMemory Queue. If I am called, I will do a reset_cache to the OS then return with the new bytes free count.

Fixed a bug in Cache_Del_Vol. The cache manager was accessing memory via a pointer, that had not been dereferenced.

High Sierra FST

Cleared invisible bit of file flags in find file so that if call is get_file_info and file is a directory, the bit will not be set.

Fixed calculation of open_buffer in setup_open_buf so it uses byte size length from directory record instead of word.

Changed build_fcr so that it sets the high bit of access word indicating that the file is never modified.

Changed build_fcr so that alloc_fcr uses pointer to volume name rather than full path, and to set bit 14 of the fcr_access word if the file is the resource fork.

Changed id_disk to return unknown_volerr when a volume is too small instead of drvr_bad_block error.

Changed get_file_type to ignore version # in file name.

Changed shutdown to respect warm_cold_flag and do nothing if warm shutdown.

Changed get_option_list to return FST id and correctly handle small buffer conditions.

Fixed get_option_list so if there is an error and the call is open, the FCR will be released.

Modified search_dir and find_file so it will return path_not_found or file_not_found errors appropriately.

Modified find_directory to return a dir_error if directory has an XAR.

Added support for version 2 of the Apple Extensions to ISO 9660.

Fixed process_path to return res_not_found instead of file_not_found when looking for resource.

Fixed bug in search_dir which caused it to fail incorrectly in certain boundary conditions.

SCSI Manager

New for System Disk 5.0.

SCSI Drivers

Many new features have been added, most notably the support for reading and writing to those devices that were formatted using 532 byte blocks and full caching support of both single and multi-block requests.

The HD Driver now offers support for removable hard disks.

A caching bug has been fixed that caused problems when the cache would not allow the addition of a deferred block.

Made changes to the HD Driver making it restartable for better performance when switching from P8 back to GS/OS.

Code has been optimized to speed up large requests that come from the cache.

Bug fixed. If there were x number of drives of the same type (i.e. 2 Hard Disks), and the first x-1 drives contained a total of x mountable partitions, then the last drive would be ignored.

Made Driver more tolerant of SCSI Devices that do not follow the SCSI Spec and the Apple Common Command Set for SCSI Devices.

Fixed problem dealing with new partitions coming online after startup check the warm start flag and not completing the initialization of the DIB.

Char FST

No Changes

Console Driver

Minor bug fixes.

Scrolling has been sped up by 2x.

Init Mgr

A message is now put up while a device is being initialized and is removed when the initialization is complete.

The Init Mgr now checks to make sure that the .CONSOLE device exists before attempting to use it. If the device doesn't exist, an error is reported.

The Init Mgr no longer rounds up to whole Megabytes when displaying the size.

AppleDisk5.25 Driver

A status call to get the configuration parameter list now returns a transfer count of $00000002.

AppleDisk3.5 Driver

This driver is restartable from memory. Now reads disks at effective 1:1 interleave on large transactions. Read and write with request count of zero does not access disk. Driver now supports access of 524 byte blocks. Shutdown call sets interleave to 4:1. ReadTrack and seek now allow interrupts when a seek moves the read write head to a new cylinder.

UniDisk3.5 Driver

This driver is restartable from memory.

Ported generated driver startup and shutdown tasks from generated driver to loaded driver. Startup call downloads and executes disk switch support code. Shutdown resets the internal hooks, effectively removing the disk switch support patch.

Fixed missing disk switch on status call $0000. Both insertion and ejection are properly reported now.

Bank0 Dispatcher

Implemented new system call $2036 D_RENAME.

Calls to devices supporting the ProDOS firmware protocol no longer validates the block number prior to passing control to the target firmware. The firmware validates the block number.

Status calls to Pascal1.1 firmware protocols now returns with bit 5 of the general status word set if a character is pending. BASIC protocols always return bit 5 regardless whether a character is pending (because we can't tell).

Device Dispatcher

Device specific cases in generated driver startup and shutdown calls no longer support the UniDisk3.5 or AppleDisk3.5 drivers. This functionality has been moved to the loaded drivers supplied for these devices.

Devices in the device list are initially marked as offline. This prevents devices that really are off line from posting a call to swap out and cache del vol which forces a search for VCR's that have not been allocated. The net effect is that ICONS mount on the desktop much faster during the boot sequence or when quitting from ProDOS8. Generated driver for AppleTalk slot now assigns a generic network device ID.

Device Manager

D_calls to device $0000 now return error $0011 = Invalid_Dev_Num.

D_status and D_control allows a status/control list pointer of zero on calls that do not use a parameter list.

Status call $0000 is now post processed so that both insertion and ejection events are posted to the OS event queue.

SCM

Fixed some bugs in set prefix code.

Busy Flag is incremented and decremented at the beginning and end of OS calls making it impossible to get into CDA when OS is busy.

Added two preference bits to sys_prefs: bit 14 controls which dialog the mount_message uses (0=normal, 1=no cancel button), bit 13 controls suppression of error dialogs (dialogs with only 1 button; 1=suppress).

Changed default system preferences to enable the mount volume message.

Modified save_screen and restore_screen to switch in text page 1 before save or restore.

Changed os_shutdown so it closes any open files after posting the shutdown event; this will ensure that the disk is clean.

Added logic to avoid doing volume_change notification on close or flush calls when no data was written.

Added d_rename call which is passed directly to the device manager. Added network_error to bouncing call #'s (is_bouncer).

Clear unclaimed interrupt counter is only cleared at startup; if it ever rolls over, we go to system death.

Added get_std_ref_num call.

Modified xlate_path so if is partial path & prefix 0 is null, will use prefix 8.

Added get_ref_info call.

Changed dispatch_signal so will always dispatch all signals regardless of interrupt state.

Added code to lock all of GS/OS managed memory on call entry and unlock it on exit.

Changed get_prefix to not force upper case on class 0 calls.

Added close_all_files routine which does end_sessions and close all with system_level at zero; forces mount volume dialogs to have no cancel button. Modified os_shutdown to call close_all_files.

AppleTalk

Added new commands:
• FILHooks ($37). Allows drivers to intercept server messages.
• FILogin2 ($38). Adds server name and zone for display in server messages. Adds AFP version number.
• FIListSessions2 ($39). Adds server name and zone.
• FIGetSVersion ($3A). Returns AFP version number set by FILogin2.
• CancelTimer ($45). Allows a timer to be cancelled, and the completion routine called.
• NBPKill ($46). Allows asyncronous NBP calls to be cancelled.
• PMCloseSession ($47). Allows an RPM session to be closed before session timer expires.

In the PMSetPrinter call, if Timeout Interval is set to zero (0), the session will never time out and must be stopped with the PMCloseSession call.

All AppleTalk driver files were moved from the SYSTEM.SETUP directory to the DRIVER directory. They now reside in files ATP1.ATROM and ATP2.ATRAM. Two new GS/OS drivers were added (ATALK & SCC.MANAGER). ATALK contains the loaded drivers .APPLETALK, .RPM (for printing through RPM), .AFP1 through .AFP14. SCC.MANAGER provides serial port arbitration and assigns unit numbers for AppleTalk device drivers.

Added support for ROM version 3 slot settings for AppleTalk.

Fixed bug in ROM patch where LAP was writing 64K bytes instead of 0 bytes. Added extra parameter checking for LAP Writes. Added extra parameter checking for PMSetPrinter. Fixed bug in PMSetPrinter so that the call will no longer accept bad input.

Server messages (connection lost, server shutting down, etc.) are displayed as dialog boxes when GS/OS is active.

PFI now sets long naming mode as the default.

PFI now supports the invisible attribute for files (in ProDOS 8; the AppleShare.FST implements this feature for GS/OS).

Fixed beep/flash server notification bug in PFI/P8 that was garbling the screen.

Fixed bug in PFI for P8 file level access.

The P8 MLIACTIVE flag is now set whenever PFI (BF00) is called.

Changed ONLINE call in PFI to force all volume names to upper case.

Fixed timer initialization bug in ATROM patch for ROM01 machines where timers installed during or shortly after initializing the protocols could complete earlier than intended.

Fixed bug in PAP where a asynchronous PAP open failing caused a concurrently pending synchronous PAP status call to hang.

ATINIT

No Change

ATResponder

Now uses the GS/OS console driver instead of built-in routines.

Responder now cancels any unknown type of request. It no longer crashes on an error from AppleTalk.

AppleShare.FST

New for System Disk 5.0.

Network Booting

New for System Disk 5.0.

Aristotle Update

New for System Disk 5.0.

QuickLogoff

New for System Disk 5.0.

AppleShare Logon CDEV

New for System Disk 5.0.

DirectConnect CDEV

New for System Disk 5.0.

AppleTalk ImageWriter CDEV

New for System Disk 5.0.

AppleTalk LaserWriter CDEV

New for System Disk 5.0.

AppleTalk ImageWriter LQ CDEV

New for System Disk 5.0.

Sys.Resources

New for System Disk 5.0.

Toolbox

System Patch: Tool.Setup

This file no longer stays in memory so it no longer is allocated low where it could create an island.

Added support for TS3. The file will not load if TS3 is missing.

System Patch: TS3 (Patch for ROM Version 03)

New for System Disk 5.0.

System Patch: TS2 (Patch for ROM Version 01)

When the bell vector is patched, we make the old value a public variable so it can be unpatched.

Fix bug in CDA interrupt handler. The original interrupt handler in ROM does a PHP, runs a bunch of code and then PLB. This is not good so, we've patched it out.

The DisposeAll code in the ROM checked for disposing of yourself, but the error broke a few applications. So now we longer return an error we just don't do anything if you try to dispose of yourself. The patch now does the same thing (nothing).

Patched rgnOp vector to better deal with low memory situations.

Tool Locator Version 3.0

Added MessageByName to tool locator.

Added the calls StartupTools and ShutdownTools.

Changed TLTextMountVolume and TLMountVolume so that it uses GetOSEvent instead of GetNextEvent. GetOSEvent doesn't call the Desk Manager. This is so these calls work when there is a system window beneath the message being displayed. Before we would lose keystrokes to the NDA beneath us.

TLTextMountVolume now sucks up and discards mouse up/downs. Before mouse up/downs were being queued up if the user was pressing the mouse while in the text screen. These mouse events would come back to haunt us when we finally went back to the super hi-res screen.

SetTSPtr now works right if the user TPT is not pointing to ROM.

Message center now allocates memory with attributes of 0.

UnloadOneTool now range checks the tool number before unloading.

Memory Manager Version 3.0

Memory Manager enhancements:

1. The algorithm for allocating memory has been optimized.

2. The OOM queue is supported. The following calls have been added: AddToOOMQueue and DeleteFromOOMQueue

Purgeall works with an ID of 0.

DisposeAll does not let you dispose of yourself. No errors are returned for compatibility reasons.

Misc Tools **Version 3.0**

Heartbeat tasks now polls MIDI before each task, and once on exit.

Added calls AddToQueue, DeleteToQueue, GetCodeResConverter, GetInterruptState, SetInterruptState and GetIntStateRecSize.

GetVector and SetVector calls now do no error checking.

The ID tag manager is now returning errors for invalid IDs.

QuickDraw **Version 3.1**

Added handling of 32 byte patterns in 640 mode.

Added support for fonts with strikes larger than 64K.

Added support for drawing in shadowed memory by setting bit 15 of master SCB on QDstart.

Sped up all object drawing commands tremendously.

Sped up text drawing in Shaston 8 with FastFont.

Used much faster algorithm for PtInRgn (also does not use any new handles).

Fast drawing routines will not be loaded on 512k machines.

OpenRgn now returns the correct error code.

bit 14 of masterSCB now signals quickdraw that the application will always use quickdraw calls whenever changing a grafport value.

Fixed bug for non-interrupt drawing of mouse.

Fix cursor vanishing bug. The cursor update routine that is called at mouse interrupt time was not clearing the proper scan line interrupt.

Desk Manager **Version 3.1**

CDA menu now scrolls. No practical limit to number of CDA's that can be installed. Number of DA's total allowed in system is 250.

TaskRunQueue implemented.

The handle to the table of CDAs was not getting unlocked. This has been fixed.

Fixed bug of clicking in drag region of an NDA and mouse getting 'stuck' if you clicked too quickly right after.

Event Manager **Version 3.0**

The keyboard interrupt routine now can translate the incoming keystrokes based on a translation table it finds in memory or in the system resource file. It uses battery RAM location $5A to decide which table to use. 0 is no table, $FF is the standard macintosh location and any other is the id of a resource in the system resource file.

GetOSEvent was disabling interrupts for too long. We now poll MIDI in 4 different places.

Scheduler **Version 2.0**

No Change

Sound Tools **Version 3.1**

Added 4 New Tool Calls, see ToolBox update for description.

Two bugs fixed. First, when tools were used in swap mode and sound had only two buffers, the second buffer was not played. Second, when playing in stereo with the sound not balanced, the last buffer full of sound would play on the wrong channel.

Fix bug in start playing. If parameter was equal to 0 then it would get stuck in an infinite loop.

ADB **Version 2.1**

No Change

SANE **Version 2.2**

No Change

Integer Math **Version 3.0**

Fixed bug in Long2Dec. Long2Dec stopped converting as soon as all the bits in the low 3 bytes zero.

Text Tools **Version 3.0**

Fixed bugs in returning errors in InitTextDev.

Tool Files

Window Manager **Version 3.1**

Added TaskMasterKey, TaskMasterContent and TaskMasterDA, CompileString, NewWindow2, ErrorWindow .

WindowManager supports same desktop drawing features as Finder (using message in message center).

Added an operation number FindDeskPatt (8) added to Desktop call. No additional parameter passed and no parameter returned. This operation makes the Window Manager check the Message Center for a DeskMessage it can use to draw the desktop image. If a message is found it will be used when drawing the desktop in the future. This call will not redraw the desktop.

Optimized VisRgn calculations. Only windows above the window brought forward have their VisRgns re-calculated.

DragRect draws 4 lines rather than 1 rect. This keeps the cursor from blinking on and off in some cases.

AlertWindow supports handles and resources.

SizeWindow and ReSizeWindow notifies controls that the window has changed size.

The window manager now zeros user memory before allocating a window record in it.

WindDragRect no longer draws to the right and below the designated rectangle.

Window titles can no longer appear past the right side of the zoom box.

PinRect now works properly with negative numbers.

CloseWindow and SelectWindow has been optimized.

TaskMaster now treats scroll bars specially for track controls in the content region. The part code returned from TrackControl is ignored since these actions cannot be cancelled by moving the mouse out of the "hot" area.

Fix bug in CloseWindow. If an application allocated storage for itself when creating a new window, the close window call might change the grafport.

Fixed bug in CloseWindow. If the WindowGlobal flag was non zero, CloseWindow was not drawing the frame properly.

Make GetFrameColor work with handles and resources.

Changing the window origin would leave an extra region allocated. It is now disposed.

Menu Manager Version 3.1

Added support for shadowing and outlined text in menus. Changed internal menu item data structure to do this and added support for the S and O characters in the parser.

Menu Shutdown now checks to see if it is active before shutting down.

Menu Manager supports scrolling menus. This is automatic. When a menu has too many items in it to be seen all at once, the menu will scroll.

Pop-up menus have been implemented.

The menu record has been modified slightly. The fields FirstItem and NumOfItems, each a byte in length, have been combined to form a word value. This value is now called NumOfItems and holds the number of items in the menu. This value is calculated when the menu item list is parsed during the NewMenu call.

The following calls have been added: NewMenuBar2, NewMenu2, InsertMItem2, SetMenuTitle2, SetMItem2, SetMItemName2, HideMenuBar, and ShowMenuBar.

Not enough memory was being allocated for the menu cache which caused menu frames and drop shadows to become corrupted when drawn. This has been fixed. (BRC #38017)

Fixed bug in GetMItem. The high word of the pointer to the item's text was not being returned on the stack correctly.

The first two bits of the menuRes flag are now being used: 00 = menu title is a ptr, 01 = menu title is a handle, 10 = menu title is a string ID.

Empty menus are now supported.

FixMenuBar now computes the width of the apple logo correctly if it gets zeroed.

Defined another bit in the menu flag. Bit 8 is defined as the ALWAYSCALLMCHOOSE flag. When set the custom defProc's mChoose routine gets called even when the mouse is not in the menu's rectangle. (It sends off the last draw message, if there is one, before calling mChoose)

The default starting position of the first menu now starts 10 pixels in for 640 mode and 5 pixels in for 320 mode. (This is to accomodate the network "busy arrow")

Keyboard equivalents and check marks will appear in plain text regardless of the style of the menu item.

Fixed bug in menu caching. Occasionally we would cache extra garbage in the menu's drop shadow.

Fixed bug where the handles to menu caches that had been purged were not getting properly disposed of.

Control Manager Version 3.1

Added super control support.

Added calls NewControl2, FindTargetControl, MakeNextControlTarget, MakeThisCtlTarget, CallCtlDefProc, NotifyControls, SendEventToControl, GetCtlID, SetCtlID, GetCtlMoreFlags, SetCtlMoreFlags, GetHandleFromID, InvalCtls.

The following types of controls have now been implemented: StaticText, Picture, IconButton, LineEdit, TextEdit, PopUp, List, GrowBox.

Added support for keystroke equivalents to buttons.

Added support for parameter text substitution with StatText items using CompileText. The global param text pointer is initialized to zero. Two calls to get and set the pointer are added.

It was possible in TrackControl to leave an extra mouse up event in the queue. Fixed.

Resource Manager Version 1.0

New for System Disk 5.0.

QuickDraw Aux Version 3.0

Added calls SeedFill and CalcMask.

The text buffer is now scaled dynamically for larger fonts.

Speed up special rectangles.

Bit 3 in fontflags word is now the Justification Flag. If 0 -> don't justify the text, if 1 -> justify. Text justification is performed only when drawing the picture.

Developer wants ClosePicture to return an error when an error had occurred while recording the picture to indicate that the picture is not valid. We do this by putting $FFFFFFFF in the pic save field if an error occurs while recording. That way, when we try to go through our normal shutdown, ClosePicture will return an error. We also fix the picture up when the error occurs so that the picture handle won't crash the system if someone trys to draw it.

Fixed a bug in picture accumulation that occurred when a ppToPort source pixel map occurred near the end of a bank.

Print Manager Version 3.0

The following calls were added: PMSetDocName, PMGetDocName, PrGetPrinterSpecs, PrGetPgOrientation.

PrChooser now puts up a dialog that says to use the control panel to select a printer.

Added code in PMGetNetworkName, PMGetUserName and PMGetZoneName to check if AppleTalk is selected before returning a string. If direct connect, then a NIL pointer is returned.

Line Edit Version 3.1

Added the control defProc for lineEdit controls.

LineEdit now supports password entries by using the high bit of the max size field to indicate whether the real string should be echoed.

Dialog Manager Version 3.1

The defProcs for StatText, LongStatText and Icon items dereferenced the control handle even when there was no handle to dereference. This caused random reads and writes to memory through out the system. They no longer do this.

Fixed bug where GetNextDItem only returned the low word of the resulting handle. It also failed when the end of the list was reached.

The dialog manager now ignores the incoming setting of the WAP. We found applications would forget to shut it down and the next time it was started up, working applications could not handle the error being reported. So rather than let good applications die, we just have the dialog manager orphan any data it has and restart.

Scrap Manager Version 3.0

No Changes

Standard File Version 3.0

Standard File has been completely rewritten to take advantage of GS/OS and support GS/OS pathnames. All old calls work as they used to.

The call SFAllCaps has been disabled and no longer has any effect on any Standard File calls.

The following calls have been added: SFShowInvisible, SFReScan, SFGetFile2, SFPGetFile2, SFMultiGet2, SFPMulitGet2, SFPutFile2, SFPPutFile2.

Note Synthesizer Version 1.4

NoteOn was trashing memory when using the 14th generator. Memory allocation at startup was off by 2.

Noteoff no longer trashes memory.

Memory was not getting disposed on shutdown.

Note Sequencer Version 1.4

Added call StartSeqRel.

Added command 30 which will call any specified routine. The routine is specified by using the databank value at startup time as the bank and the word of data as the address. No registers are preserved, and the routine is called at interrupt time.

StopSound was not cleaning the stack correctly if a StartSeq had not been made.

Font Manager Version 3.1

Larger fonts are now supported and are limited at this point by the memory size.

Fixed bug relating to purge status of derived fonts.

ChooseFont no longer clears all update events from queue before starting.

Fixed bug in routine ReadFamList. An incorrect buffer was being used with a file READ call. This would cause the Font Manager to crash on occasion.

Version number for new font format is version 1.5.

List Manager Version 3.1

List controls are now supported.

Changed way SelectMember works so that if the item is already visible, the list is not moved. If the item is 1 off the bottom, the list is scrolled by 1.

ListMgr can handle C-strings when the list control is created using NewControl2.

Bit 5 of the flags byte of ListMember record now means never select this item.

The following calls have been added: DrawMember2, NextMember2, ResetMember2, \ SelectMember2, SortList2.

TextEdit Version 1.0

New for System Disk 5.0.

ACE **Version 1.1**

No Changes

MIDI Tool Set **Version 1.3**

Now supports _MidiClock miSetFreq with values up to $FFFF.

The sound tools must be started up before the MIDI tools. If they are not, a WAP error is returned from the Sound Tools and this error is now passed on by MidiStartup.

Fixed midi.control emptyoutbuf/discardoutbuf. Calls now return with Accumulator = 0

Midi.Device no longer trashes random memory.

Print Drivers

LaserWriter Version 3.0

Driver now checks for the case where the height of the image is negative (this can happen if the imagewidth is nonzero but all the bits in the font strike are 0s) and marks it as undefined so as not to print garbage.

Jobname is now set in the LaserWriter's status dictionary using information from SetDocName so AppleTalk can use it to report status. The message now appears as: "job: User - user name, Document - doc name; status: busy; source: AppleTalk." PrCloseDoc resets the name to the default string "Unknown", making the name valid for the duration of the job only.

Driver can print using PostScript fonts that are downloaded in the printer. Note: downloading needs be done by a host running HFS or MS-DOS.

Two new calls are added: PrGetPrinterSpecs, which returns the printer's type and its characteristics (only color capability is defined for now) and PrGetPageOrientation which returns either portrait or landscape. A new header is added to denote this as a driver which support the new calls (and any subsequent new calls). This header is 2 words long: first word is 0 followed by a call count word.

The values for the horizontal resolution is changed from 42 to 40 in 320 mode and from 84 to 80 in 640 mode. The incorrect values cause margin sizes to be wrong.

All file calls are converted from ProDOS to GS/OS class 1. Added ability to create multiple files for PostScript output using Command-F. The files will be named PostScript.GSxx where xx is 00-99 numbered sequentially in the order created.

Bitmap font downloading code is modified to use pointers to the various parts of the font record on QuickDraw's zero page. This is necessary due to changes in the Font Manager - font strike larger than 64K and in QuickDraw - FastFonts.

QD verb erase now fills with background pattern or color. Previously, white is used to fill.

Foreground color is now used as a solid pattern for text printing. Previously, the pen pattern was used for text.

PicComments can now be greater than 255 bytes in length.

In RoundRects, the OvalWidth and OvalHeight parameters were swapped. They are now used in the correct order.

Frames of objects no longer have part of the frame be outside the boundary of the shape.

PrOpenDoc now just look at the type and version field of the print record instead of calling PrValidate.

Constants for Hres and Vres are adjusted so the page sizes and margins are more accurate.

ImageWriter Version 3.0

Two new calls are added: PrGetPrinterSpecs, which returns the printer's type and its characteristics (only color capability is defined for now) and PrGetPageOrientation which returns either portrait or landscape. A new header to denote this as a driver which support the new calls

(and any subsequent new calls). This header is 2 words long: first word is 0 followed by a call count word.

Driver now checks the size of Quick Draw's buffer if Better Text is requested. If the buffer is too small, we grow the buffer to what we need and restore it back to the original size.

Status display code now uses LETextBox2 instead of DrawString, so text will be wrapped, especially for 320 mode.

Band buffer allocation now uses as much memory as available, up to 64K.

PrOpenDoc now just looks at the type and version field of the print record instead of calling PrValidate.

The newly added ClosePicture error is now passed to the application.

ImageWriter.LQ Version 3.0

Two new calls are added: PrGetPrinterSpecs, which returns the printer's type and its characteristics (only color capability is defined for now) and PrGetPageOrientation which returns either portrait or landscape. A new header to denote this as a driver which support the new calls (and any subsequent new calls). This header is 2 words long: first word is 0 followed by a call count word.

Driver now checks the size of Quick Draw's buffer if Better Text is requested. If the buffer is too small, we grow the buffer to what we need and restore it back to the original size.

Status display code now uses LETextBox2 instead of DrawString, so text will be wrapped, especially for 320 mode.

Band buffer allocation now uses as much memory as available, up to 64K.

PrOpenDoc now just looks at the type and version field of the print record instead of calling PrValidate.

The newly added ClosePicture error is now passed to the application.

Epson Version 2.0

Two new calls are added: PrGetPrinterSpecs, which returns the printer's type and its characteristics (only color capability is defined for now) and PrGetPageOrientation which returns either portrait or landscape. A new header to denote this as a driver which support the new calls (and any subsequent new calls). This header is 2 words long: first word is 0 followed by a call count word.

Driver now checks the size of Quick Draw's buffer if Better Text is requested. If the buffer is too small, we grow the buffer to what we need and restore it back to the original size.

Status display code now uses LETextBox2 instead of DrawString, so text will be wrapped, especially for 320 mode.

Band buffer allocation now uses as much memory as available, up to 64K.

PrOpenDoc now just looks at the type and version field of the print record instead of calling PrValidate.

The newly added ClosePicture error is now passed to the application.

Epson printers do not have paper reverse capabilities, so the code to reverse the paper is removed and printing will occur on the current line. Also, the LX model does not support absolute position of the print head, so movement to the right is simulated with spaces. The old code calculated the number of spaces based on the left margin instead of at the end of the last printed string thus causing large gaps to appear. Any movement to the left is ignored and will print from current horizontal position.

ImageWriter Emulator   Version 1.3

Memory problems when printing large documents is resolved by adding code to save and restore virtual memory used after each page.

Two new procedures are added which uses the stack to transfer necessary variables across save and restore pairs.

In EscDict, 3 procs were defined using the incorrect decimal equivalent of the command: ESC- should be 45, not 44; ESC> should be 62, not 61; ESC? should be 63, not 62. Also the comment for ESC+ and ESC- are swapped. ESC g (103) is defined to use procedure gr3*4; but in the actual definition of the procedure, the name is defined as g3*4. Changed dict to use g3*4.

popCmdDict is added in ignore since all commands which use ignore needs to pop the dict off the stack.

Removed check for prop? in gf since that flag has effect only on text and not graphics. Graphics only need the dpi value.

Removed gsave and grestore in gprint. Graphics were being printed on top of each other since the current point is always restored to where the first graphic call begins.

Port Drivers

AppleTalk   Version 3.0

A new header is added to denote this as a driver which will support new calls. This header is 2 words long: first word is 0 followed by a call count word.

Removed 3 calls, GetZoneList, GetPrinterList and GetMyZone, that are no longer used since the Choose Printer functionality has been moved to the Control Panel.

AppleTalk PAP for printers is designed so that when an open is received, it waits 2 seconds for other opens, if any, and then accepts the one which has been waiting the longest. During this time, the printer will not accept any other type of PAP commands. The user may decide to cancel the print job at this early stage, so a 2 second wait loop is added to disallow closing of the connection during that window.

Printer   Version 2.0

A new header is added to denote this as a driver which will support new calls. This header is 2 words long: first word is 0 followed by a call count word.

Buffering in the firmware is enabled so we can read the self ID response from the printer. There is problem with handshaking with the printer if the buffer is turned off so we left it on. Problems exists where the reset for the next page happens before last part of the page is finished, losing the remaining data. Code is added to query the output queue to make sure it is empty before the next firmware reset. Also added a flag to skip the query if the firmware has not been initialized.

The firmware is reset after printing so the firmware's buffers will be deallocated. Also added code to remember and reset the output redirection back to the way we found it.

Modem **Version 2.0**

A new header is added to denote this as a driver which will support new calls. This header is 2 words long: first word is 0 followed by a call count word.

Buffering in the firmware is enabled so we can read the self ID response from the printer. There is problem with handshaking with the printer if the buffer is turned off so we left it on. Problems exists where the reset for the next page happens before last part of the page is finished, losing the remaining data. Code is added to query the output queue to make sure it is empty before the next firmware reset. Also added a flag to skip the query if the firmware has not been initialized.

The firmware is reset after printing so the firmware's buffers will be deallocated. Also added code to remember and reset the output redirection back to the way we found it.

Parallel.Card **Version 2.0**

Changed driver to be as general and non-assuming as possible to support other parallel cards. There may be cards that cannot be set such that they do not try to interpret $89's as a special control character. If there is such a card, then part of the buffer will be sent, then the control character will be changed from an $89 to an $81, and then the process will be repeated until an $81 is encountered. This process is repeated until the buffer is entirely sent. After the buffer is sent, then the special character is set back to an $89. The "recommended way" to ID Pascal and BASIC devices is:
Pascal I.D bytes CN05=38 CN07=18 CN0B=01 CN0C=ci
Apple parallel intf card: CN05=48 CN07=48

A new header is added to denote this as a driver which will support new calls. This header is 2 words long: first word is 0 followed by a call count word.

Midi Drivers

Apple.Midi Version 1.2

The main interrupt is no longer disabled when the SCC is reset.

Card6850.Midi Version 1.1

Minor changes.

Finder & Utilities

Note: A thumbnail sketch of improvements to the Finder & Utilities suite is provided here. For more detailed information, please refer to the *Apple IIGS Owners' Reference* or the appropriate ERS.

Advanced Disk Utility Version 1.1

ADU now has a custom desktop icon.

You may now use ADU to create more than 7 partitions per SCSI card. The new limit per SCSI *device* is

MAX(MIN((DriveSizeInBytes DIV (32*1024*1024)), 32),8)

where MIN gives the lesser of its two arguments, and MAX the greater. In other words, you may always create at least eight partitions per device, but never more than 32 under any circumstances. However, if the size of your device does not permit 32 partitions of 32MB each, you will be limited on that device to a number of partitions equal to the number of complete 32MB partititions that the device can hold. Note that several devices, each with its maximum number of partitions, can be connected to a single SCSI card.

The OK button in the info dialog now highlights when selected by RETURN key.

Fixed bug in which Edit menu was offset by one: Paste became Copy, Copy became Cut, Cut did nothing, etc.

What you see is now (reliably) what you get in terms of partition size. (In the past, ADU had, under some circumstances, rounded the size of a partition to the next full Megabyte at initialization.)

Fixed a problem with the partitioning code. Under some circumstances, ADU thought that disks were larger than they really are.

Erase, Initialize, Zero, and Partition are now dimmed when the startup volume is the current volume. You should not be able to obliterate the startup disk with ADU.

The bar graph behaves properly when users select new partition after resizing a partition.

If your startup disk is a partition, we no longer permit you to partition the device on which the startup "disk" resides. ADU no longer hangs when getting info on a CD Drive, the first partition of which is offline.

ADU was not shutting down the tool locator correctly, thus leaving RAM tools in an unpurgeable state. Since the tools could not be purged, there was insufficient memory for the Finder, which would post an appropriate error message before giving up. That problem has been fixed.

Clicking on the partition name in the Get Info dialog when there is only one partition will no longer cause the information to blink.

Given the DIB mechanism in GS/OS, there is no way to distinguish temporarily ejected partitions from permanently removed ones. Both are marked as "offline," but remain in the DIB chain until the next system reboot. This affects two situations in particular: 1) when you repartition a hard disk, the DIBs for old partitions that are not part of the new partitioning layout remain in the system as "offline" orphans; and, 2) when you eject CDROM partitions or drag them to the trash, the corresponding DIBS are marked as "offline," until such time as you reinsert the disk, which you may never do. Without a great deal of analysis, which is imprudent to add to the program at this stage of development and might not be totally reliable in all cases anyway, ADU cannot deduce which offline partitions are truly "dead" and which are "merely restin'." So, it now ignores all offline partitions in its Info scan, and shows only the active ones. This is by design. The bottom line: if you threw a CD ROM partition in the Trash using the Finder before launching ADU, you shouldn't be able to see that partition within ADU, either.

Finder **Version 1.3**

Finder 1.3 focuses primarily on network awareness and the new Icon Info menu item (which replaces the Get Info item of Finder 1.2). Menus and the menu bar have been rearranged slightly (i.e., some items missing, some have new names, some are in new menu-bar items, such as "Disk," etc.). Information on the desktop, in folder windows, and in the Icon Info displays is updated dynamically wherever appropriate and practical.

Icon Info uses a new, Info-card metaphor. Size, kind, and modification date information is given by the General card, location information by the Where card, network access privileges and owner information by the Access card, and commentary by the Comment card. Not every card is displayed for every icon. For instance, Comment cards are now only shown for AppleShare objects, because the ProDOS file system does not support the attachment of commentary to files. Also, Access cards are only shown for AppleShare volumes and folders, since notions of privilege and owner only apply to those objects.

In the General card and elsewhere in the Finder, if object names are too long to fit in the space reserved to display them, the ends are truncated and replaced with the ellipsis (...).

Icon Info General Card includes a "Contents" field and a graphic "calculate" icon. The Icon Info General card for a disk or network volume now always displays the coarse size of the volume (figured using the "blocks free and in use" information provided by the operating system). The "Calculate" button does not apply to the Size fields of Icon Info General cards for disks or volumes. You may press "Calculate," however, to learn the number of files and folders on a disk or volume.

More room is provided for "largish icons" in Icon Info windows and on the desktop (i.e., the "cleanup" grid is now larger).

The Icon Info Comment card shows any comments that are attached to an AppleShare file, but the user may not edit the comment.

Trash Can's Icon Info General card no longer includes either lock check box or Size field.

The Finder tries to preload the icon button resource before attempting to display Icon Info. The user is given two chances to insert the system disk before the Finder decides that the preload has failed. If the preload fails, the Finder informs the user that the startup disk must be present for the operation to continue, before giving one more chance to try again or cancel.

The Rename and Remove menu items are gone.

Now, you need only single-click on an Icon's name field to rename that icon. In general, the Finder does not let you enter file names that exceed 31 characters in length. If you attempt to exceed the limit (while renaming an Icon, say), the overflowing character is NOT accepted and the system beeps at you (or blinks the menu bar if the sound volume is turned all the way down). Characters typed up to the overflow point are preserved. We did this rather than putting up a dialog in the spirit of "Demonstration, not Conversation."

Finder now supports object names of up to 31 characters in length, in addition to lower-case characters, punctuation, and the Mac-like graphic symbols. Finder accepts both upper case and lower case letters in ProDOS file names, by virtue of accommodating the corresponding new feature of the ProDOS FST. Also, the first character of an icon name need not be a letter. Of course, the Finder will still reject a file/path name that contains illegal characters for the file system in which the target object resides (or will be placed). However, the error condition will only be noted (by alert dialog) when the name is actually used in a GS/OS call and an error is returned by the appropriate FST. Please note that the Finder remains insensitive to case when comparing file/path names (such as when trying to determine if source file names are already present on destination volumes, or when checking to see whether the name of a newly-inserted disk conflicts with that of a currently-mounted disk).

New folders are automatically selected for renaming and are placed in available window space (perhaps off the screen). Note that the space that would be occupied by invisible files or by files that are on the desktop is not considered as being "available."

Duplicates are offset 45° to the lower right of their originals.

CDEVs join the class of files that can be "Inactive," and generic load files (type $BC) have been excluded.

Because window manipulations can be so much faster now, we have increased to 10 the number of windows that can be open simultaneously on the desktop.

The "talking man," "caution sign" and "stop sign" icons are now used in Finder dialogs, instead of the custom Finder alert icons used in 4.0.

The Lock icon is now used to indicate locked status of iconic objects on the desktop and in folder windows (whether viewed by icon or text).

The horizontal scroll bar is disabled when directory windows are viewed textually.

In general, ESC key functions as the keyboard equivalent of Cancel in Finder dialogs.

We are now using the term "Disk" instead of "Volume" throughout the Finder's dialogs.

Finder now respects the Message Center's Picture message.

Changed the way the launching message is displayed. We now use the .CONSOLE driver instead of the Text Tools so the Finder does not depend on the state of slot 3 anymore.

For reasons of compatibility with older Finders, we cannot save a desktop icon's "owner" pathname in the FINDER.ROOT file (where desktop state info is stored) if that pathname is longer than 65 characters. While it is now possible to drag deeply nested objects to the desktop, leave them there, and work with them (i.e., drag them to the trash) during the current session, icons whose complete pathnames exceed 65 characters will be automatically "Put Away" whenever the Finder is shut down or an application is launched. The user must retrieve those icons by hand whenever Finder execution resumes. Icons with full pathnames that do not exceed the 65 character limit will remain on the desktop until 1) they are moved by hand into a folder on the disk upon which they reside; 2) they are Put Away; 3) the (now normally invisible) FINDER.ROOT file is removed or damaged.

Read-only operations in the Finder now open files for read-only access. This will help keep us "network friendly" and "resource friendly."

Whenever possible, larger buffers are now used (and so, better performance is achieved) during file and disk copy operations.

The Initialize and Erase dialogs no longer have default buttons. According to the human interface guidelines, the default action should be the least harmful (in this case, "Cancel"), and we concur. However, power-users were rightfully confused that a tap of the RETURN key after they had entered a new name for the initialized or erased disk caused ejection of the target medium! In the interests of safety and sanity, not to mention consistency with the human interface guidelines, we have eliminated the default behavior. You must now explicitly click on "Initialize/Erase" or "Cancel." The Finder ignores the RETURN key in these cases.

We have achieved a relatively seamless transition between the (now-"desktop-blue") boot-thermometer screen and the Finder desktop. Thanks to the GS/OS team for their cooperation in making this effect possible.

Erasing, Initializing, or overwriting a disk that has open files (e.g., the Startup Disk) is no longer permitted. The Finder posts a dialog to inform the user that the requested operation can not be completed.

If you insert a disk with the same name as a currently mounted disk that has at least one open file (e.g., the startup disk, while the system resource file is open), the Finder will now alert you to the name conflict and give you an opportunity to rename the disk if it is a valid ProDOS disk. If not, the Finder posts a generic "disk name conflict" alert and permits you to eject removable media if possible.

Eject menu item is now disabled when icon is dithered (i.e., when the corresponding item is offline).

We have eliminated the "Help for dimmed menu commands" preference and mechanism. The old "Resize RAMDisk" preference has migrated into the Graphic Control Panel's RAM CDEV, where it belongs. "Hide Finder's data files" has become "Hide invisible files" (since Finder data files are now invisible).

Finder icons that contain colored elements (i.e., portions that are not black, white, or grey) have been changed so as not to accept coloration via the Finder's Color menu. This does not apply to user-defined icons, however. In particular, you cannot color the icons for RAMDisk, hard-disk drive, or hard-disk partition, nor application icons for Advanced Disk Utility or Installer.

The Finder no longer closes or collapses windows during the launch of an application program.

We have partially addressed a problem with backward compatibility to 1.1 and 1.2 Finders. The 5.0 Finder was saving uppercase and lowercase names for icons into its FINDER.DATA and FINDER.ROOT files. Earlier Finders may sometimes become confused about icon placement when using such data files, in particular because they cannot associate the lower-or-mixed case icon names with otherwise identical uppercase file names provided by ProDOS-16 and the first release of GS/OS. Now, the Finder coerces all alpha-character information to uppercase before saving it within FINDER.DATA files. Note, however, that the Finder does not similarly coerce alpha information within FINDER.ROOT. Although this can lead Finders 1.1 and 1.2 to display some desktop icons as lower-or-mixed case, to the best of our knowledge, the Finder shouldn't (and doesn't) engage in any destructive or embarrassing behavior when presented with lowercase characters from FINDER.ROOT.

The Finder now disables its desktop refresh routine upon exit, which avoids a horrible system death in case anyone has clobbered low-lying zero-page globals.

Clipboard text scrolling now works correctly. We are using a better rectangle definition when clipping the contents of the window.

Copying a disk to another disk of different size forces the destination window closed as the end of the copy.

Whenever filename translation is necessary during a copy operation, the core design of the Finder does not reliably permit the user to skip the copying of files that are nested within folders that the user selected directly. In other words, it is OK to skip an item if that item corresponds to one of the icons you specifically selected on the desktop, but it is not OK to skip any children of that item. We have introduced a new filename translation dialog to handle the latter case. This is the same as the original translation dialog (which still appears in the former case), except that the buttons for skipping and translating all files automatically have been removed. The user must elect to translate or cancel the copy operation.

In order to help optimize RAM usage in minimum-configuration systems, we have modified the Finder's icon mechanism somewhat. The old FINDER.ICONS file is now two files, one called FINDER.ICONS, and the other, FINDER.ICONS.X. The FINDER.ICONS file contains all the icons for devices that can be or contain startup volumes, and a few folder and document icons that we feel are essential to the operation of any IIGS system. FINDER.ICONS.X contains all other "standard" icons for folders, documents, devices, and applications. On a minimum-configuration system, the Finder will load its icons only from the FINDER.ICONS file on the startup volume, and will ignore all other icons files, including FINDER.ICONS.X. This saves roughly 7K of runtime RAM over the previous, all-inclusive FINDER.ICONS file.

Also to ameliorate the RAM situation on minimum-configuration systems, the Finder's Help facility and certain low-level routines associated with the About box have been moved into dynamic segments.

Finder is normally considered to be a restartable program. However, in tight-memory situations, should the zombied segments of the Finder already have been purged, the Loader will sometimes reload one or more segments into the SHR screen area. The Finder now includes code that detects such problems, and deals with them by forcing the Finder to launch (NOT restart) itself. A relaunch will purge memory and specify that Finder segments are not to be loaded into special RAM, (e.g., the SHR screen area).

The Finder tables for file kind translation have been expanded, to match the standard list of names maintained by Developer Technical Support. However, only minimum filetypes are supported for 512K systems, in keeping with our memory conservation efforts. This is accomplished by having the filetype name strings split into two separate files, FTYPE.MAIN and FTYPE.AUX. These files reside in the ICONS subdirectory, and only FTYPE.MAIN is loaded when the total memory in the system is 512K or less.

Verification does not apply to AppleShare network volumes.

The icons slashed-pencil and slashed-folder are now displayed in a server window's info-bar, as appropriate. The slashed-pencil indicates that the user has no write-access privileges in the folder; the slashed-folder indicates no see-files privileges.

Erase, Initialize, and Verify menu items are dimmed whenever a desktop selection includes one or more server volumes.

The renaming mechanism does not function for AppleShare file server volumes. That is to say, the icon title field cannot be edited. This is the same mechanism that prevents renaming of locked icons and the Trash Can.

The delay between updates of server-volume windows is 30 seconds. In order to promote timely updating of all icons in a server window, we were having to get the complete directories corresponding to open server windows. Even doing this no more frequently than every thirty seconds led to unacceptable performance when server windows containing more than a handful of icons were left open. The problem is that certain changes in a server-resident directory (e.g., file/folder lock status) do NOT update the volume/folder modification date, so the Finder can never be notified of such changes; it must always check explicitly. We have compromised by updating server windows ONLY WHEN the volume/folder modification date changes. We still have to get complete catalogs when we DO update, but on the average, we will be doing so far less often than before. Performance could be very bad on very busy systems with very full directories. In such cases, users should be advised to close windows that are not in use.

It is inappropriate at the present time for us to use the desktop database scroll-bar values, so we are now forcing all AppleShare views to the upper-left corner (i.e., both scroll bars at "zero.")

Installer **Version 1.1**

The Installer now has a custom desktop icon.

Multiple script selections are now possible. You may select any number of scripts of any type. When one or more of the selected scripts requires that the user select a folder, the folder selection window becomes active. The selected folder, referred to as the Application Folder, will be used by ALL of the scripts that require folder selection. Of course, scripts that do not require folder selection will continue to place all their files in the destination locations specified by the script. Scripts in a multiple-selection are executed in the order in which their names are displayed (i.e., in alphabetical order, from the top of the selection window to the bottom).

Can now install onto any disk including the volume from which Installer was launched.

Allows use of tab key to select between destination volumes.

The Installer now recognizes and can process files on AppleShare file servers, if you have the proper access privileges.

The amount of disk swapping necessary for single-drive operation has been cut in half.

The Installer can now handle both data and resource forks of any extended GS/OS file (storage_type = $05).

The Application Folder Selection window does not display invisible files (i.e., files with I-bit set). Only folders for which the user has the proper access privileges will be active.

You may now move the Installer's main window by clicking in the title-bar and dragging, just as with any other standard window.

The Install Everything Possible" script is gone, since you may now pick and choose the updates to install. There are now two "Latest System Files" installer scripts. The old, familiar, "Latest System Files" script is the same as it always was, and is appropriate for updating system and application disks when you intend to use the Finder as your program selector (the recommended case). The script, "Latest System Files (No Finder)" is useful when you wish to update an application disk, but do not wish to overwrite the file SYSTEM/START with the Finder. Typically, such a disk will be a turnkey application disk produced by a third party developer, and SYSTEM/START will be the application itself, instead of the Finder.

The pop-up menu is deactivated when the script does not require directory selection.

Added code to deactivate numerous controls and internal mechanisms if there are no scripts selected, a possibility that could not occur originally but is more likely now, since multiple selections are possible.

Anytime a system level operation occurs on the boot disk, ALL desk accessories—both "New" and "Classic"—are disabled and the user may no longer Quit from the Installer back to the launching application. (A system level operation is defined as an Install or Remove that does not require application folder selection.)

When the above described situation arises, the only way to leave the Installer program is to restart the system, preferably by accepting the default response that is available in the dialog box that appears when Quit is "pressed." This change was implemented to solve two potential problems:

• The SYS.RESOURCES file is open and contains resources that applications may expect to be intact. However, the Installer may have closed the file and replaced it on the boot disk.

• The Installer may have replaced tools on the boot disk. However, old RAM resident tools may still be running out of RAM expecting to be able to load compatible old tools from disk which have been replaced.

After altering anything on the boot disk you simply cannot be guaranteed that any application or desk accessory will still run properly. Restarting the system is the only way to be sure that the operating system, tools, and resources, whether on the boot disk, or RAM resident, are compatible.

Cosmetic change: Caution alert icon is now colored yellow and Stop alert is now colored red. Wherever this has made sense, all instances of the word "volume" have been changed to "disk" in all Installer scripts.

The Installer now posts a dialog to notify the user when installation and/or removal is completed.

Installer dims "New Folder" button at the ninth nesting level.

Application Folder information is displayed whenever the current selection includes an application script. Otherwise, the name of the destination disk is displayed (as appropriate for a system-level installation).

Installer now closes all NDAs when system level installation on the boot disk is about to take place.

The Installer is no longer restartable from memory, and will remain so until such time as we can discover how memory owned by the Installer was being clobbered.

Added a flag to indicate if any scripts are actually executed and eliminated the "successful completion" message if none are.

Added code to skip over AppleShare volumes that are not read and write enabled.

Installer now ignores "drop-folder volumes," since space calculations vital to the installation process cannot be performed unless the Installer has both write *and* read access to the volume.

Scripts with a ScriptVersion number of "V1.10" are now accepted along with V1.00 scripts. *YOU MUST designate your scripts with ScriptVersion "V1.10" if any of the source files have resource forks*. This will keep earlier versions of the Installer from trying to process those scripts and the associated extended files. Earlier versions of the Installer do not handle resource forks correctly, and proper use of the ScriptVersion facility will ensure that they are never asked to try.

The Installer now issues a "caution alert" before executing a script if the second character in the script header's ScriptFlag field is *lower-case*. The caution alert contains a warning icon, the text of the ScriptHelp field, and two buttons, one to cause execution of the script in question, and one to skip execution of that script. There is no default button; the user must make a decision on how to proceed. The caution alert is appropriate in situations where execution of a particular script could create an inconsistent or incomplete system, or would result in the installation of inappropriate facilities (e.g., AppleTalk drivers or FSTs on a stand-alone system).

Added a new FileSpec flag ("D"), which may only be used with a "4" flag; the script entry *must* include a CreationDate/Time. The specified file will be deleted only if it is *strictly older* than the FileSpec's CreationDate/Time. The Latest System Files and Latest System Files (No Finder) updates have been rewritten to use the new flag to remove System Disk 4.0 files whenever the user elects to Install the update.

Scripts for the following updates use the new "D" flag to rid the destination of old System Disk 4.0 files: AppleShare on 3.5 Disk, Latest Sys. Files (No Finder), Latest System Files, Local Network Startup, Server Network Startup.

Control Panel NDA Version 1.0

New for System Disk 5.0. See user documentation and/or ERS.

CDRemote NDA Version 1.2

New for System Disk 5.0. See user documentation and/or ERS.

VideoMix NDA Version 1.0

New for System Disk 5.0. See user documentation and/or ERS.

BASIC.SYSTEM Version 1.3

Fixed the problem with BSAVE that caused problems when saving a smaller file with the same name as a larger file. End of File mark is now properly updated.

Fixed the bug in Chain/Store that caused the system to bomb when exact multiples of 256 bytes of data were chained from one program to the next.

Added the "MTR" command to allow the user to enter the monitor by typing "MTR" as an option to Call-151.

July 17, 1989

Dear Apple II Developer,

This month's mailing includes a collection of Engineering Reference Specification (ERS) documents concerning the GS/OS part of Apple IIGS System Software 5.0.

These ERS documents are ***preliminary*** and ***subject to change***. While most of the information contained in these documents is technically correct, some information is neither supported nor guaranteed to exist in future versions of the software. Certain procedures may only be available to internal system software components, and using them will guarantee future incompatibility for your application.

If you find something in this documentation that looks like it may not be supported in future system software versions, contact AIIDTS before using it. Remember, compatibility is your responsibility as well as Apple's, but we cannot help you if you don't take the time to find out. If you have questions, contact us at one of the following addresses:

Apple II Developer Technical Support
Apple Computer, Inc.
20525 Mariani Avenue, M/S 75–3A
Cupertino, CA 95014
AppleLink: AIIDTS
MCI Mail: AIIDTS (264-0103)

Apple will provide updated, release documents describing all the features of GS/OS 3.0 later this summer, and at that time, you should destroy these preliminary ERS documents.

Thank you,

Apple Computer, Inc.

TABLE OF CONTENTS

An Introduction to Universe GS/OS

External ERS

by
Lee Collings

Version 1.0
March 22, 1989

Introduction to Universe GS/OS Update

The new system disk contains many enhancements that will improve the way the user perceives the performance of the Apple IIGS. This introduction is designed to offer you a quick overview to many of the more important changes that have occurred in the GS/OS operating environment. Detailed descriptions are included in the the ERS and Delta Documents.

1. GS/OS Switching

GS/OS System disk 5.0 now supports quick switching between P8 and GS/OS. When a P8 program is launched from a GS/OS native environment (such as the Finder), GS/OS will be stored in memory so that returning to GS/OS from the P8 environment is very fast.

2. New ExpressLoad (Quicker Application/Code Loading)

ExpressLoad is designed to accomplish quicker loading of applications and load files. (load times can be improved by 200 to 500 percent!) In order for the file to be loader by ExpressLoad the application or load file must be stored in ExpressLoad format. A conversion utility is available in both MPW and APW formats to allow developers to convert to the ExpressLoad format. Note that a file stored in ExpressLoad format is still compatible with the standard system loader.

3. New SCSI Manager and SCSI HD & CD Drivers

The new SCSI Manager and Drivers offer a wide range of support for SCSI devices. These all new segments offer up to a 4x increase in SCSI system performance. Separate drivers have been created for HD and CD ROM environments allowing for increased flexibility. The SCSI Manager will allow for future SCSI drivers to be quickly defined and added as they are developed and will also allow for future SCSI peripherals to be easily added to the IIGS system. These new drivers take full advantage of the GS/OS Caching mechanism ans also support both single and multiple block I/O which decreases the hardware overhead.

4. New and Enhanced System Calls

Two new Class 1 calls designed to add and delete notification procedures for the new General Notification Queue have been added to GS/OS. This queue mechanism will allow applications to be notified when certain OS events occur.

A new prefix "@" has been defined to help applications be more "AppleShare" aware. This prefix is set whenever an application is launched and is set to the folder where the application lives, unless the application is being launched from a file server. When launched from a file server the prefix is set to use the users folder defined on the server.

Two new preference bits have been defined to enhance the control of OS dialogs along with numerous other calls for the OS Environment. Also Pathname to Reference Number and Reference Number to Pathname calls. Plus a special mechanism which keeps disks from being Erased or Formatted if file are open on the disk.

5. Enhanced Interrupt and Signal Management

A Signal dispatching queue has been defined to offer the best possible method for handling signals sent to GS/OS. The Signal handler determines whether or not interrupts are enabled and how to react.

6. GS/OS Managers

Initialization Manager - No Major Changes

Cache Manager - System Disk 5.0 contains a new cache manager that adds an additional feature. The new feature, referred to as **AutoFlush**, allows better use of a small cache when sessions are enabled.The Cache Manager now places itself in the "OutOfMemory" queue. If the memory manager cannot allocate the requested memory the Cache Manager will purge the GS/OS cache.

Device Manager - Performance significantly improved for applications using D_xxx calls. Added system call D_Rename allowing renamable devices to be renamed.

Device Dispatch - Performance significantly improved for device access. Support for notification of disk switched events (poll driven) through the OS notification manager has been added. Alert dialogs are displayed for devices that require a driver that is not presently installed on the system disk. Support for keeping GS/OS in memory while running ProDOS 8 applications has been added.

7. GLoader/GQuit Changes

Second Entry Point added to GLoader - A new entry point has been added to GLoader to offer support for AppleShare Boot code or other boot code that has a need for this newly defined entry point. Custom Boot Flags are defined by the Boot code are used by GLoader to customize the booting process.

New Prefix Support - Prefix "@" has been added to GS/OS for AppleShare support and is set by GQuit before launching a GS/OS application. The "@" prefix is set to a pathname specified by the AppleShare FST when the application being launched resided on an AppleShare Volume.

OS Switching - Major changes have been made to the switching mechanism to allow for quicker switching from P8 applications to GS/OS. When a P8 program is launched from the GS/OS environment, (such as the Finder) GS/OS is stored in memory thus allowing for a very quick return to GS/OS when the P8 application is exited.

Additional changes have been made in the GQuit /GLoader program segments to add or increase support for ProDOS 8 compatibility, 512K System support, Application loading, ExpressLoad support, Notification Queue support, Boot Driver support, Init Files, FST Installation and I/O Redirection.

8. Redirection Support 10,11,12

Support for prefixes 10, 11 and 12 (STDIN, STDOUT & STDERROR) have been defined as Standard I/O prefixes to allow for increased flexibility in I/O redirection.

9. GS/OS Drivers

AppleDisk 3.5 - The AppleDisk 3.5 driver is now restartable and does not need to be loaded off disk when returning to GS/OS after running a ProDOS 8 application.

AppleDisk 5.25 - No Changes

UniDisk 3.5 - The UniDisk 3.5 driver is now restartable and does not need to be loaded off disk when returning to GS/OS after running a ProDOS 8 application.

Console Driver - Two new calls have been added to the Console driver as well as performance enhancements that allow better throughput to the screen. The two calls that have been added are Add Trap and Reset Trap. Performance for the Console driver is about eleven times faster for single character writes. Enhancements to the Console driver include faster screen scrolling and faster write operations. The console driver is now restartable and does not need to be loaded off disk when returning to GS/Os after running a ProDOS 8 application.

10. GS/OS FST (File System Translators)

ProDOS FST - The new version of the FST now writes all dirty blocks to disk when the last file that has been modified is closed. This means that files opened for read access do not affect when the bitmap and directory blocks are updated. This new feature is very important since the GS resource manager leaves the system resource file open until the system is rebooted!

Another new feature of the ProDOS FST is it's ability to add a resource fork to a standard file. This new feature allows files to be retrofitted with a resource fork. Additionally, the ProDOS FST uses the cache manager more often which translates to better performance. Also the option list is now used by the ProDOS FST.

Support for notification of volume change events through the OS notification manager has also been added.

High Sierra FST - Added support for Version 2 of Apple Extensions to ISO 9660.

Character FST - No Changes

Apple Extensions to ISO 9660[1]

Version 2.2
Bryan Atsatt & Brian Bechtel

Who Should Read This Document

- Apple Developers working with ISO 9660
- Publishers of authoring tools for ISO 9660 discs
- Publishers of ISO 9660 discs
- Publishers of ISO 9660 receiving system software

Release History

1.0	7/7/88	Initial release.
1.1	7/14/88	Removed reference to Macintosh in footnote. Modified note in ProDOS Filename Transformation section to require that Volume Identifier be transformed.
1.2	7/22/88	Changed Protocol Identifier to all uppercase.
1.3	8/2/88	Added SystemUseID 06 for Macintosh Finder flags.
1.4	8/13/88	Corrected location of RecordPad field so comes before the SystemUse field. Added padding fields to SystemUseID's 2-5.
2.0	3/6/89	To accomodate other receiving systems using the SystemUse field, the identification has been slightly modified. A new signature has been added, with some unused fields in the Macintosh extensions removed.
2.1	4/6/89	Described when the Protocol Identifier is required and how to use the extensions on CD-ROM XA discs.
2.2	5/30/89	Corrected SystemUse format description for HFS (SystemUseID 02).

Introduction

It may be desirable to create an ISO 9660 CD ROM containing HFS and/or ProDOS files in order to benefit by the storage capacity and distribution cost savings of CD ROM, and the interchange advantages of ISO 9660. However, both the HFS and ProDOS file systems require information that the ISO 9660 file system does not support: ProDOS requires a file type and an auxiliary file type, while HFS requires a file type, a file creator, and file attributes.

This document defines a protocol which extends the ISO 9660 specification to include HFS and ProDOS specific information, without corrupting the ISO 9660 structures. Discs created using the protocol are valid ISO 9660 discs, and should not behave differently on non-Apple receiving systems.

In addition to preserving file specific information, the protocol defines a mechanism for preserving filenames across the ProDOS → ISO 9660 → ProDOS translation.

The protocol was designed to solve existing compatibility problems as well as allow for future expansion. It uses the SystemIdentifier field in the Primary Volume Descriptor for global information, and the SystemUse field in the directory record for file specific information.

[1] ISO 9660 is the international file system standard, based on the "High Sierra" standard, for CD ROM. The protocol defined in this document is intended for the ISO 9660 file system; however, it is also supported for the High Sierra file system.

The Protocol Identifier

Location:	SystemIdentifier field in the Primary Volume Descriptor
Size:	32 bytes
Contents:	"APPLE COMPUTER, INC., TYPE: " followed by a long word type. In hex:

41 50 50 4C 45 20 43 4F 4D 50 55 54 45 52 2C 20 49 4E 43 2E 2C 20 54 59 50 45 3A 20
3x 3x 3x 3x

Type: The last four bytes of the identifier contain nibble encoded type information. Nibble encoding is necessary in order to guarantee that these are legal ISO 9660 "a-characters" (i.e. printable characters). The type bytes are numbered 0-3; type(0) is the byte following the space ($20). The bits of each type byte are numbered 0-7, 0 being the least significant. The type bytes are defined as follows:

type(0):	bit 0	1 = Perform ProDOS filename transformation (described later)
	bit 1-3	reserved, must be zero
	bit 4	must be 1
	bit 5	must be 1
	bit 6	must be 0
	bit 7	must be 0
type(1):	bit 0-3	reserved, must be zero
	bit 4	must be 1
	bit 5	must be 1
	bit 6	must be 0
	bit 7	must be 0
type(2):	bit 0-3	reserved, must be zero
	bit 4	must be 1
	bit 5	must be 1
	bit 6	must be 0
	bit 7	must be 0
type(3):	bit 0-3	Apple Extensions version number (2 indicates this version)
	bit 4	must be 1
	bit 5	must be 1
	bit 6	must be 0
	bit 7	must be 0

The identifier is considered valid if the first 28 bytes match. The identifier is required only for global information; it is not currently required for the Macintosh or for the //GS if the ProDOS filename transformation is not needed under GS/OS.

The Directory Record SystemUse Field

Directory records in the ISO 9660 specification have the following format:

byte	DirectoryRcdLength
byte	XARlength
struct	ExtentLocation
struct	DataLength
struct	RecordingDateTime
byte	FileFlags
byte	FileUnitSize
byte	InterleaveGapSize
long	VolumeSequenceNum
byte	FileNameLength
char	FileName[FileNameLength]

byte	RecordPad (if necessary)
char	SystemUse[SystemUseLength]
byte	RecordPad (if necessary)

The RecordPad field is present only if needed to make DirectoryRcdLength an even number. If present, the RecordPad field ***must*** be zero ($00). The SystemUse field is an optional field under ISO 9660; it is defined for our use below.

The SystemUse field, when present, must begin with a signature word , followed by a one byte length and a one byte SystemUseID, followed by file system specific information. This structure may be repeated multiple times for multiple file systems; however, the total length of the SystemUse area (SystemUseLength) ***must*** be an even number. The signature word allows a receiving system to ensure that it can interpret the following data correctly, and the SystemUseID determines the type and format of the information which follows.

CD-ROM XA discs must be handled as a special-case in that the signature word, if present, will begin at byte 14 of the SystemUse field rather than at byte 0.

There are two AppleSignatures. The preferred AppleSignature is defined as "AA" ($41 41). The old AppleSignature, used for a previous version of this document, was defined as "BA" ($42 41). Disks pressed using the old format are still supported, but we recommend that new disks be pressed using only the current format.

Receiving systems must perform a simple calculation to determine if the SystemUse field is present in any given directory record. It is present if:

DirectoryRcdLength - FileNameLength > 34

Receiving systems should first verify that the SystemUse field is present (making sure to account for the possibility of a RecordPad field), then check for the AppleSignature before interpreting the SystemUseID.

The SystemUseID is defined as follows for an AppleSignature of "AA":

SystemUseID	Definition
$00	reserved.
$01	ProDOS file_type and aux_type follow.
$02	HFS fileType, fileCreator, finder flags
$03-FF	reserved.

The following tables define in detail the data formats for each SystemUseID. The MSB-LSB notation ("MSB" = Most Significant Byte, "LSB" = Least Significant Byte) means that the MSB occupies the lowest address, while LSB-MSB means that the LSB occupies the lowest address.

SystemUseID 01, ProDOS:

SystemUse offset	Contents
$00-01	$41 41 (AppleSignature)
$02	SystemUse Extension length ($07 for this ID, includes AppleSignature bytes)
$03	$01 (SystemUseID)
$04	ProDOS file type
$05-06	ProDOS aux type (LSB-MSB)

Note that a padding byte must be added if this ID is the only extension added to the directory record.

SystemUseID 02, HFS:

SystemUse offset	Contents
$00-$01	$41 $41 (AppleSignature)
$02	SystemUse Extension length ($0E for this ID, includes AppleSignature bytes)
$03	$02 (SystemUseID)
$04-$07	HFS fileType (MSB-LSB)
$08-$0B	HFS fileCreator (MSB-LSB)
$0C-$0D	HFS finder flags (MSB-LSB)

The authoring software can simply copy the finder flags as retrieved by the HFS call PBGetFInfo. Only bits 5 (always switchLaunch), 12 (system file), 13 (bundle bit), and 15 (locked) are used. All other bits are either ignored or set due to internal workings of the file system translator. See Macintosh technical note #40 for more details about the finder flags.

The Extension to ISO 9660

This section describes the extension to ISO 9660 which allows multiple users of the "System Use" field. All references are to ISO 9660, First edition, 1988-04-15.

Section 9.1.13 System Use [PB (LEN_DR - LEN_SU + 1) to LEN_DR] shall be replaced as follows:

This field shall be optional. If present, this field shall be reserved for system use. If necessary, so that the Directory Record comprises an even number of bytes, a (00) byte shall be added to terminate this field.

If this field is present, it shall be broken up into a series of System Use Extensions. There can be more than one System Use Extension for a given directory record. A System Use Extension shall have the following format:

BP 1 : byte 1 of the signature. This field shall contain an 8-bit number. This field shall be recorded according to 7.1.1.

BP 2 : byte 2 of the signature. This field shall contain an 8-bit number. This field shall be recorded according to 7.1.1.

BP 3 (LEN_SE): This field shall specify as an 8-bit number the length in bytes of this System Use Extension. This field shall contain an 8-bit number. This field shall be recorded according to 7.1.1. This field shall include the length of the two signature bytes preceeding this byte.

BP 4 to LEN_SE : this field shall be reserved for system use. Its content is not specified by this International Standard.

Multiple System Use Extension fields may exist for a given directory record, subject only to the limitation that the total length of a directory record must be able to be recorded in the 8-bit field defined in section 9.1.1.

NOTE: For CD-ROM XA discs, the Byte Positions listed above must be adjusted upwards by 14 to account for XA's 14 fixed length system use field.

The ProDOS Filename Transformation

This section defines a mechanism which preserves ProDOS filename syntax. Legal filenames in ProDOS differ from legal filenames under ISO 9660:

- ProDOS filenames allow multiple periods; ISO 9660 does not.
- ISO 9660 requires that the two separators "." and ";" exist and that the ";" be followed by a version number. This requirement is for files only, not directories.

This requires a transformation of some sort. Fortunately, this specific combination of file systems allows us to define a reversible transformation which the authoring tool and receiving system can use to hide the fact that a transformation has occurred.

When creating an ISO 9660 disc from ProDOS source files, the authoring tool must perform the following transformation on *all* filenames:

- Replace any periods (".", $2E) with underscore ("_", $5F).
- If the filename refers to a directory, the transformation is finished.
- If the filename refers to a file, append the characters ".;1" ($2E, 3B, 31).

Examples:

ProDOS Filename	Type	ISO 9660 Filename
PRODOS	file	PRODOS.;1
BASIC.SYSTEM	file	BASIC_SYSTEM.;1
SYSTEM	directory	SYSTEM
DESK.ACCS	directory	DESK_ACCS
START.GS.OS	file	START_GS_OS.;1

NOTE: The Volume Identifier in the Primary Volume Descriptor is a filename, and, therefore, must also be transformed. It must be transformed as if it is a directory *name.*

When the ProDOS Transformation bit is set in the Protocol Identifier, the receiving system can handle the necessary conversions such that the *original* ProDOS filenames can be used to refer to all files and directories on the volume:

- Before searching the disc for a given pathname, perform the transformation described above.
- Before returning any names found, reverse the transformation above:
    - Strip ".;1" if present.
    - Replace any underscores ("_", $5F) with periods (".", $2E).

This process must be done for every file and directory on disc.

Guidelines For Transforming HFS Filenames

Since most HFS filenames are illegal in the ISO 9660 specification, some transformation will be required when creating an ISO 9660 disc with HFS files. No reversible transformation is possible without degrading performance; therefore, we can only define guidelines for publishers and authoring tool publishers to follow when performing the transformation:

- convert all lowercase characters to uppercase.
- replace all illegal characters, including periods, with underscore ("_", $5F).
- if the filename must be shortened, truncate the rightmost characters.
- if the filename refers to a file, append the characters ".;1" ($2E, 3B, 31).

Following these guidelines will result in more consistent discs.

ISO 9660 Associated Files

Associated files are exactly analogous to resource forks. Though the format of associated files is clear in the ISO 9660 specification, we would like to re-state it here:

> An associated file is defined as having the associated bit set in the file flags byte of the directory record. It has exactly the same file identifier as its counterpart, and resides ***immediately before*** its counterpart in the directory. The associated file is treated as the resource fork, its counterpart is treated as the data fork of the file.
>
> For example, if a file "FOO.;1" has an associated file, there will be two adjacent directory records named "FOO.;1"; the first one (the resource fork) will have the associated bit set, the second one (the data fork) will have the associated bit clear.

The Macintosh File System

The Apple Macintosh file system is named "HFS", for "Hierarchical File System." This file system is similar to both MS-DOS and Unix in supporting subdirectories, where files may be logically grouped together. Subdirectories on the Macintosh are called folders.

When the Macintosh was first introduced, a flat file system called "MFS" for "Macintosh File System" was used. The MFS file system did not adequately provide for high capacity storage media such as hard disks and CDROM, so the HFS format was introduced in its place. MFS format is now generally used only on 400K, single sided floppy disks. HFS is compatible with MFS.

Macintosh file and volume names are different from both Unix and MS-DOS in that they allow any character except the colon ":" as a valid file name character. This means that a name such as "My payroll file" is perfectly acceptable on the Macintosh. There is no concept of a file name extension, such as is found in the MS-DOS or High Sierra formats. HFS does not distinguish between upper and lower case names; the names "forecast", "Forecast", and "FoReCaSt" all refer to the same file. Volume names may have a maximum of 27 characters, and file names may have a maximum of 31 characters. To specify a file name in a folder (subdirectory), you give the folder name, followed by a colon, followed by the file name. For example, if you have the file "My payroll file" in the folder "business", you could specify the file as "business:My payroll file". To specify a file in the folder immediately above the current folder, you may use the shortcut name of two colons, "::".

An area where HFS is different from MS-DOS or Unix is in the concept of file forks. A file has two forks; a resource fork and a data fork. The data fork is used for the same purpose as a "file" in MS-DOS or Unix; it contains information in an unstructured format. The data fork is used by an application to store the contents of the document.

The resource fork of a file contains Macintosh resources. Resources are data in a special format; Apple Computer has defined many "famous" types of resources, and we have provided facilities so that programmers may define their own custom resource types. The resource fork of a file is accessed using special program calls, from a part of the Macintosh operating system known as the Resource Manager. All resources are labeled with a resource type of four characters. Some examples of resources are: 'CODE', for executable code; 'ICON' for a 16 bit by 16 bit icon; 'ICN#', a slightly different icon resource used by the Finder to display an unique icon for a file; 'MENU' for the menus found at the top of every Macintosh application; and 'STR ', for strings used within a program. There are many other types of resources as well.

Resources provide the Macintosh with a flexibility not found in other operating systems. For instance, to "localize" a well-written application into another language, all that needs to be changed are the resources that are language specific (usually the 'MENU' and 'STR ' resources.)

Apple provides developers with several tools and resource editors that assist in modifying resources.

As well as the resource and data forks, additional information is kept about each file. This information is used by the Finder to display the position, attributes, and icon of a file. The additional information consists of a file type of four characters (such as 'TEXT' for plain text or unknown type documents, 'APPL' for applications, 'MACA' for MacWrite documents, and so forth), a file creator unique to each application (such as 'MWRT', 'EXCL', 'FNDR') and file attributes (such as invisible, system file, or locked against accidental deletion.) The additional information also includes a mapping of file creator to appropriate application, so that a Macintosh user may select a document and open it, and the appropriate application will be invoked and passed the document name.

Problems with High Sierra Support on the Macintosh

Apple supports both High Sierra formats through the use of a feature in the Macintosh file system called the "external file system" hook. This is a special low-memory global which contains a pointer to a list of external file system handlers. For High Sierra support, Apple has written a new set of routines contained in a file called "Foreign File Access". This file, plus the files "High Sierra File Access" and "ISO 9660 File Access" combine to provide complete support for the High Sierra formats. Apple's High Sierra software supports level 2 interchange, according to section 10.2 of the ISO 9660 specification. This means that Apple does not yet support interleaved files or multi-disc volumes.

The Apple extensions do not fully correct deficiencies in the High Sierra specification; some issues are handled by the High Sierra File Access software. For instance, in HFS, the Finder keeps track of the position of a file on the desktop or in a folder by using a special field inside HFS; under High Sierra, a file's position is computed when the folder is opened, and can not be changed.

Because of some deficiencies in the original design of the Finder, it is not possible to have the correct icons display for a file on a High Sierra disc. This is because the Finder does not actually ask for the icon of a file; rather, it assumes the existence of a *desktop database* which contains these mappings, and makes a special call, giving only the file creator and type. The software to provide this information was designed to be very HFS specific. Apple Computer will address this defect in a future release of the operating system. Currently, even if a file's icon bitmap is defined in the Apple extensions, it will not be used by the Finder. All files that reside on a High Sierra disc will display a generic icon. If such a file is copied to a hard disk, the correct icon will be displayed. You can still double-click on a High Sierra file, and it will open correctly.

If a High Sierra disc has been pressed without the Apple extensions, all files on the disc are considered to be of type 'TEXT' and creator 'hscd'. Type 'TEXT' is a generic type which can be read successfully by many Macintosh applications.

There has been some confusion about padding bytes in a directory record which also contains Apple extensions. There are two possible places in a directory record where a padding field may occur: after the File Identifier, and at the end of the directory record. If no Apple extensions exist for a file, then the two locations are identical, and the rule stated in section 9.1.12 applies: "...(a padding field) shall be present ...only if the number in the Length of the File Identifier is an even number." If Apple extensions exist, then section 9.1.13 *also* applies: "If necessary, so that the Directory Record comprises an even number of bytes, a (00) byte shall be added to terminate this field." (i.e. the System Use field used for Apple extensions.)

Sample Code for Retrieving Macintosh File Information

This code is an example of how to access the Macintosh file type, file creator, and finder flag information. The code is written in the C language.

```
/*********************************************************************
 *
 *  Function:         GetFileInfo
 *
 *  Purpose:          get lengths of file rsrc and data forks, type, creator,
 *                and finder flags
 *
 *  Returns:          OSErr
 *                noErr, unless PBGetFInfo has an error.
 *
 *  Side Effects:     rsrcLength, dataLength, fType, fCreator, flags are
 *                    updated with correct values for the file requested
 *
 *  Description:  call PBGetFInfo() and return its results.  This routine
 *                will only work if the path name to the file on the Mac
 *                can fit in 255 characters (the length of a pascal
 *                string)  See Inside Macintosh, Volume 2, page II-115
 *                and Mac Tech Note #40 for more information.
 *
 *********************************************************************/
OSErr
GetFileInfo(name, vRefNum, rsrcLength, dataLength, fType, fCreator, flags)
StringPtr     name;
short         vRefNum;
long          *rsrcLength;
long          *dataLength;
OSType        *fType;
OSType        *fCreator;
short         *flags;
{
    HParamBlockRec    io;
    OSErr             result;

    io.fileParam.ioCompletion = 0L;
    io.fileParam.ioNamePtr = name;
    io.fileParam.ioVRefNum = vRefNum;
    io.fileParam.ioFVersNum = 0;
    io.fileParam.ioFDirIndex = 0;
    result = PBGetFInfo(&io, false);
    if (result == noErr)
    {
        *rsrcLength = io.fileParam.ioFlRLgLen;
        *dataLength = io.fileParam.ioFlLgLen;
        *fType = io.fileParam.ioFlFndrInfo.fdType;
        *fCreator = io.fileParam.ioFlFndrInfo.fdCreator;
        *flags = io.fileParam.ioFlFndrInfo.fdFlags;
    }
    else
    {
        *rsrcLength = 0L;
        *dataLength = 0L;
        *fType = 0L;
        *fCreator = 0L;
        *flags = 0;
    }
    return result;
}
```

GS/OS AppleShare File System Translator
External ERS

Version 0.19

By Mark Day

Revision History

Date	Version	Who	Description
11/2/87	0.01	Mark Day	Preliminary release. Limited distribution.
11/24/87	0.02	Mark Day	Filetype equivalence section rewritten. The server does support Read/Write/Destroy/Rename/Backup bits. Other typos and errors corrected.
1/29/88	0.03	Mark Day	Byte Range Lock supported through FST_SPECIFIC call. Specil Open Fork supported through FST_SPECIFIC call. Format of option_list proposed.
5/2/88	0.04	Mark Day	General Implementation section re-written. READ not supported for directories. A plea for deny modes. Filetype equivalence section updated and table added. FST specific call "Cache Control" added. Changed definition of Lock Flag in Byte Range Lock. Added implementation details for Change Path.
6/7/88	0.05	Mark Day	Firmed up some areas where there were questions.
7/12/88	0.06	Mark Day	Cleaned up some confusing areas (especially in the FST_SPECIFIC call). FST_SPECIFIC will use a single parameter list (the two parameter list proposal has been rejected).
7/27/88	0.07	Mark Day	"Cache Control" turned into "Buffer Control" to be more accurate. Option_list structure better defined. Documented the preferred method of enumerating directories.
8/16/88	0.08	Mark Day	Create's EOF and resource EOF fields are ignored. Removed comments about Set_Prefix being passed to an FST. Open defaults to buffering on. Special Open Fork defaults to buffering off. Set_EOF and Get_EOF force writes of dirty buffers. Read_Block is not supported at the application level. Get_FST_Info corrected to return file_sys_id of $0D for AppleShare. Discussion of Read with newline moved to Read call (was in buffer control). FST_SPECIFIC commands re-ordered; GetPrivileges, SetPrivileges, Map Name, Map ID, User Info, and Copy File commands added.
8/17/88	0.09	Mark Day	Corrected constants for Byte Range Lock in Appendix A. Close always removes the FCR even if there is an error.
8/24/88	0.10	Mark Day	Get Privileges, Set Privileges, User Info now take G-Strings instead of user and group IDs. Map Name and Map ID have been removed (User Info and Copy File have different command numbers now).

8/30/88	0.11	Mark Day	Added section on pathname syntax. More specific error code information given. Removed reference to option_list in CREATE. SET_FILE_INFO only uses Finder Info from option_list. Folder's read bit in access word clear if no see files, no see folders. File not found will be distinguished from path not found. Class 0 and class 1 OPEN calls work differently with respect to deny modes. Parameters to Byte Range Lock corrected.
10/20/88	0.12	Mark Day	Corrected error in description of UserInfo command. Expanded information about error codes for FST_Specific commands. Documented Read across byte range locks. Directories have 2K buffers.
10/27/88	0.13	Mark Day	More complete description of how buffering works (see the description of Buffer Control). Added definition of null strings for owner/group names in Get/Set Priveleges. File_sys_id added to beginning of option_list. Added desktop and comment calls. Now an external ERS.
11/30/88	0.14	Mark Day	Inserted FST_Specific call GetUserPath before desktop calls. Added discussion about making calls with interrupts disabled.
12/14/88	0.15	Mark Day	CloseDesktop takes a volume name in addition to the desktop refnum. Beware: the format of the OpenDesktop and CloseDesktop parameter lists may change (the calls may even disappear). Added GetSrvrName call.
1/10/89	0.16	Mark Day	Open with pCount > 4 returns error $4E if you can't see the object. Added error codes for User and Group Name not Found.
3/6/89	0.17	Mark Day	Added diagram of option_list info. In option_list, the ShortName has been replaced by Access Rights. Changed error codes for block-level commands. Remember, the access word returned by OPEN is NOT an indication of the access you got when the file was opened (see discussion of OPEN call).
4/25/89	0.18	Mark Day	A little more detail about the format of the Finder flags in the option_list (courtesy a Macintosh tech note). Struck reference to the FST-specific call OpenDesktop going away.
5/11/89	0.19	Mark Day	UserInfo returns null string for user name if logged on as guest, and null string for primary group if the user has none. Fixed diagram of the option_list (the minimum buffer size and actual data size fields were wrong).

Changes are marked with a change bar in the margin.

Table of Contents

Introduction

This document describes the implementation of the AppleShare File System Translator for GS/OS. It assumes a familiarity with GS/OS and AppleTalk. Refer to the AppleShare IIGS Programmer's Guide for information about AppleTalk protocols. Please note that AppleShare for GS/OS encompasses not only the AppleShare FST, but also the AppleTalk protocol stack, drivers, network booting (if desired), switching between the GS/OS FST and the ProDOS 8 PFI (since AppleShare will be accessible from both ProDOS 8 and GS/OS), and enhancements to the Finder to make it network aware.

The AppleShare FST is the implementation of AppleShare for GS/OS. It is meant to supercede AppleShare IIGS (Toucan), the implementation of AppleShare for ProDOS 16. Since ProDOS 16 makes calls to ProDOS 8 to get its work done, Toucan patches the ProDOS 8 MLI to intercept calls bound for the network. In this way, both ProDOS 8 and ProDOS 16 can use network volumes. GS/OS is completely separate from ProDOS 8. The ProDOS 8 MLI will still be patched to intercept network calls while ProDOS 8 is running. When GS/OS is running, GS/OS will make calls directly to the AppleTalk routines via the AppleShare FST, instead of calling ProDOS 8 to make the AppleTalk calls. This will increase the speed of GS/OS programs using files on the network (compared to ProDOS 16).

The AppleShare FST will only work with file servers supporting AFP version 2.0 or greater.

Compatibility

An important consideration for the AppleShare File System Translator is backwards compatibility with ProDOS 16 and ProDOS 8 implementations of AppleShare (namely projects Bullwinkle and Toucan). All documented calls that were added to the ProDOS 8 MLI to support AppleShare will still be usable from ProDOS 8. The RamDispatch vector at $E11014 will continue to support full native mode calls from either ProDOS 8 or GS/OS. In addition, device-specific calls will be included to allow uniform access to the AppleTalk protocols and file system features such as Byte Range Lock.

The class 0 Open call will work as the Open call for Toucan (AppleShare for ProDOS16) did. The class 1 Open call is more restrictive in its setting of deny modes which is safer for opening files. Please use class 1 Open whenever possible, and try to use the requested access parameter when possible (eg.: only ask for read if that is all you will do with the file).

File not found and path not found errors will be reported correctly (when a file is not found, the FST will check for the existance of the parent and issue a path not found if the parent does not exist). This differs from ProDOS16 which reported path not found for both path not found and file not found error conditions.

Pathname Syntax

There are two kinds of syntactic restrictions on pathname syntax: those imposed by GS/OS, and those imposed by the FST (because of naming restrictions in AFP).

GS/OS may impose a maximum length on pathnames. The AppleShare FST does not.

The span of a pathname is the maximum number of characters in a filename (i.e. between pathname separators, including volume names). GS/OS imposes no restriction on maximum span. The AppleShare FST restricts the maximum span to be less than 32 characters. While AFP volume names are less than 28 characters, this part of the syntax is not checked. Volume names with a length of 28-31 will return a volume not found error.

GS/OS allows "/" or ":" to be a separator. The first "/" or ":" in the pathname is taken to be the separator. A ":" can never be used in a filename. A "/" cannot be used in a filename if the separator is "/". The AppleShare FST disallows a null byte in a pathname. All other characters are permitted. Note that the high bit of a character is significant. Characters with values greater than or equal to 128 are considered extended ASCII and typically display as special symbols on Macintosh and IBM systems.

Numbers as the first filename in a partial pathname are assumed by GS/OS to be prefix designators. Since numbers are valid filenames in AFP, a prefix designator should always be used explicitly with partial pathnames beginning with a number. For example, "0:555:Hello" refers to a file "Hello" in a folder "555" relative to prefix 0; "555:Hello" will give an invalid path syntax error since GS/OS assumes that "555:" is a prefix designator for prefix 555, which is invalid.

Equivalence of Macintosh and GS/OS File Types

AppleShare file servers supporting AFP version 2.0 or greater maintain both Macintosh filetype and creator as well as ProDOS filetype and auxtype. Since the filetype information for the two operating systems are distinct, a workstation can set one kind of filetype for Macintosh and another type for ProDOS.

The AppleShare FST will use the ProDOS filetype and auxtype fields; it depends on the server to derive appropriate type information for Macintosh files. The AppleShare File Server version 2.0 uses a convention also used by Apple File Exchange and the MAX cross-development tools.

ProDOS files are distinguished by a Macintosh creator of "pdos". The ProDOS filetype SYS (=$FF) has a Macintosh filetype of "PSYS". The ProDOS filetype S16 (=$B3) has a Macintosh filetype of "PS16". The ProDOS unknown filetype (=$00) has Macintosh filetype "BINA". ProDOS text files (TXT = $04) with auxtype of $0000 (i.e. normal ASCII text, no records) has Macintosh filetype "TEXT". These special cases allow Macintosh to display unique icons for these filetypes.

Macintosh files with creator "pdos" and a filetype of the form "XY " (two hex digits followed by two spaces) will get ProDOS filetype $XY and auxtype $0000. Macintosh files with creator "pdos" and a filetype of the form $70uvwxyz ($70 is a lower-case "p") have ProDOS filetype $uv and auxtype $wxyz (note the order of the bytes: on the Macintosh they are stored high-low instead of low-high).

APW source files (ProDOS filetype $B0) are given Macintosh filetype "TEXT" so that they can be edited more easily.

The conversion rules are summarized in the following tables. If more than one rule applies, the one closest to the top of the table will be used.

ProDOS -> Macintosh conversion

ProDOS Filetype	Auxtype	Macintosh Creator	Filetype
$00	$0000	"pdos"	"BINA"
$B0 (SRC)	(any)	"pdos"	"TEXT"
$04 (TXT)	$0000	"pdos"	"TEXT"
$FF (SYS)	(any)	"pdos"	"PSYS"
$B3 (S16)	(any)	"pdos"	"PS16"
$uv	$wxyz	"pdos"	"p" $uv $wx $yz

Macintosh -> ProDOS conversion

Macintosh Creator	Filetype	ProDOS Filetype	Auxtype
(any)	"BINA"	$00	$0000
(any)	"TEXT"	$04 (TXT)	$0000
"pdos"	"PSYS"	$FF (SYS)	$0000
"pdos"	"PS16"	$B3 (S16)	$0000
"pdos"	"XYΔΔ" †	$XY	$0000
"pdos"	"p" $uv $wx $yz	$uv	$wxyz
(any)	(any)	$00	$0000

† Where X,Y are hex digits (i.e. "0"-"9" or "A"-"F"), and Δ is a space

System Calls

This section describes differences of parameters between the AppleShare FST and the ProDOS FST. Please see the System Call ERS for more detailed information about these calls. Any calls not documented here behave as specified in the System Call ERS.

CREATE ($01)

The ProDOS filetype and auxtype will be set to the values given in the call; by default, the Macintosh creator will be set to "pdos" and the Macintosh filetype will be derived according to the rules above. All files will be created as extended files (i.e. have both a data and a resource fork) since there is no way to distinguish between a fork of length 0 and a fork that does not exist.

In a class 1 call, the EOF and resource_EOF fields are ignored. This is because the definition of the call states that the forks' EOFs will be set to 0, and it is impossible with AFP to allocate space in a fork past its EOF.

Only the low byte of the filetype and low word of the auxtype will be used. If the high byte of the filetype or high word of the auxtype is non-zero, an invalid parameter error will be returned.

SET_FILE_INFO ($05)

The ProDOS filetype and auxtype will be set to the values given in the call; by default, the Macintosh creator will be set to "pdos" and the Macintosh filetype will be derived according to the rules above. The option_list data is the same as for the GET_FILE_INFO call, except that only the Finder Info is used (the other fields cannot be set); any data past the Finder Info field is ignored.

If the file_sys_id field is not the same as AppleShare's file system ID ($0D), then the option_list is ignored. All FSTs will return their file system ID in the first word of the option_list and will ignore setting of the option_list info if the file_sys_id does not match theirs. This allows applications to always get and set the option_list as part of the copying process even when copying from one file system to another.

GET_FILE_INFO ($06)

Folders with no see files and no see folders access will have the read bit in their access word cleared; files, and folders with see files or see folders, have their read bit set. If the file's resource fork is not empty, the storage_type will be returned as $05 (extended), otherwise it will be returned as $01/$02/$03 (seedling, sapling, or tree) depending on the data fork's length. The option_list's data is structured as follows:

(This may change)

word	File_Sys_ID ($0D for AppleShare)
32 bytes	Finder Info
long	Parent Directory ID
4 bytes	Access rights (same format as Get-/SetPrivileges)

See Appendix B for a diagram of the option_list structure.

According to *Inside Macintosh IV*, the Finder Info for a file looks like this:

4 chars	Macintosh filetype	
4 chars	Macintosh creator	
word	Finder flags	
	Bit	Meaning
	0	Set if file/folder is on the desktop (Finder 5.0 and later)
	1	bFOwnAppl (used internally)
	2	reserved (currently unused)
	3	reserved (currently unused)
	4	bFNever (never SwitchLaunch) (not implemented)
	5	bFAlways (always SwitchLaunch)
	6	Set if file is a shareable application
	7	reserved (used by System)
	8	Inited (seen by Finder)
	9	Changed (used internally by Finder)
	10	Busy (copied from File System busy bit)
	11	NoCopy (not used in 5.0 and later, formerly called BOZO)
	12	System (set if file is a system file)
	13	HasBundle
	14	Invisible
	15	Locked
point	File icon's location in window word vertical word horizontal	
word	Window # file belongs to -3 = file is in Trash window -2 = file is on desktop 0 = file is in disk window	
word	Icon ID	
8 bytes	(unused)	
word	Comment ID	
long	Directory ID of home folder (for Put Away)	

For files, the Finder Info looks like this:

rectangle	Folder's window rectangle
	word top
	word left
	word bottom
	word right
word	Finder flags (same as for a file??)
point	Folder icon's location in window
	word vertical
	word horizontal
word	Folder's view (??)
point	Window's scroll position
	word vertical
	word horizontal
long	Directory ID chain of open folders (??)
word	(unused)
word	Comment ID
long	Directory ID of home folder (for Put Away)

The access rights field for directories is in the same format as used in the GetPrivileges and SetPrivileges calls. For files, the field is set to all zeros. Note: this field was included to allow applications like the Finder to determine what access a user has to a folder without having to do a separate GetPrivileges call.

OPEN ($10)

The access, filetype, auxtype, and option_list parameters are as described in the SET_FILE_INFO call. If request_access is $0000 (as permitted), an attempt will be made to open the file as read/write, deny read/write. If this fails, an attempt will be made to open the file as read-only, deny write. If this fails, an attempt will be made to open the file as write-only, deny read/write. If this also fails, an access denied error ($4E) will be returned.

If the class is 0, an attempt will be made to open the file as read/write deny write. If this fails an attempt will be made to open the file as read-only deny nothing. If this fails, an attempt will be made to open the file as write-only deny write. If this also fails, an access denied error ($4E) will be returned. This behavior is the same as for ProDOS16 and was done for compatibility with ProDOS16.

Note that using class 0 Open allows files to be opened by multiple users and does not fully prevent one user from changing data that another user is reading, but it does allow multiple users to read a file without changing existing code. Class 1 Open prevents one user from writing data that another user is reading, but does not allow multiple users to read a file without explicitly asking for read-only access. Putting a file in a folder with no make changes access will cause both class 0 Open and class 1 Open with request_access = 0 to open the file for read-only and will allow multiple users to read the file (and not allow the file to be written to).

If request_access is $0001 (read-only), the file will be opened as read-only, deny write. If it is $0003 (read/write), the file will be opened as read/write, deny read/write. If it is $0002 (write-only), the file will be opened as write-only, deny read/write. If the file cannot be opened with the requested mode, an access denied error will be returned.

If you want to open a file with permissions different than above, you should use the FST specific command "Special Open Fork". That call is essentially the same as the open command, but it lets you control all of the permission bits yourself.

The System Loader should be modified to load files by opening them Read-only, Deny Write (request_access as $0001).

By default, buffering will be turned on for files or directories opened with this call. The buffer will not be filled until the first Read or Get_Dir_Entry call is made (so that buffering may be turned off after the open but before the first read). The size of the buffer for files is 512 bytes; for directories it is 2048 bytes.

Folders with neither see files nor see folders access rights cannot be opened (since the only valid operation on an open folder is GET_DIR_ENTRY). The returned error code is $4E (access denied).

With a class 1 call, all of the parameters after the resource number are file information. Think of this as a combined GET_FILE_INFO and OPEN call (and in fact, that is how it behaves). In particular, the access word returned is not an indication of the access rights you have when the file is opened; it is really a "best case" access to the file. The actual access you get when opening the file is controlled by several things: the access word, access privileges to ancestor and parent folders, and access restrictions ("deny modes") imposed by other users who have the file open.

Note that using a class 1 open with request_access = 0, is usually not a good idea since you don't know what access you really got to the file (until you try) because the FST will try several combinations as described above. If your application can deal with several different kinds of access to the file, it is best to try those different access modes individually until you get one you can handle. For example, if you can handle either read-write or read-only access but prefer read-write, try opening the file with request_access = 3 (read-write). If this fails, try opening with request_access = 1 (read-only). This way you will know exactly what access you have to the file. Rememer, too that if you use class 1 open for read-write, nobody else will be able to open the file and multiple users won't be able to run your application at the same time.

If you use a class 1 call with pCount > 4 (i.e. you are asking for file info to be returned), and you don't have privileges to see the object you are opening (if the object is in a drop box, for example), the call will return with an error $4E (access denied), since you don't have access to get the file info you requested.

READ ($12)
The READ call will not be supported for directories. ProDOS directories will not be synthesized. One should use the Get_Dir_Entry call to enumerate directories. A read on a directory will return error $4E (access denied).

If part of the range to be read is locked by another workstation, a $4E error will be returned and the transfer count will be set to indicate the number of bytes transfered before the locked range was encountered.

Regardless of the value in the cache priority field, data will not be put in the system cache. By default the FST maintains a block buffer containing the 512 bytes of the block containing the current mark. This block buffer can be controlled on a per-file basis by the FST specific call "Buffer Control".

If buffering is disabled and newline mode has been enabled with more than one newline character, the read will be completed one byte at a time. This is done because the server's newline mechanism provides for only one newline character. Beware that this mode of reading a file imposes tremendous amounts of overhead and should be avoided if at all possible.

WRITE ($13)
Regardless of the value in the cache priority field, data will not be put in the system cache. By default the FST maintains a block buffer containing the 512 bytes of the block containing the current mark. This block buffer can be controlled on a per-file basis by the FST specific call "Buffer Control".

Writes to directories are not allowed. They will return error $4E (access denied).

CLOSE ($14)
The file will alwaysbe closed (i.e. the FCR will be removed), even if there is an error. This is because any error an application gets may not be correctable by the application or the user (eg. the data to be flushed before the close is locked by another workstation, or a connection has been lost with the server). It seems better to simply indicate the error and clean up the system as much as possible (i.e. remove the FCR, and reduce the VCR's open file count).

SET_EOF ($18)
If a fork is extended (make longer), the additional bytes will be allocated but might not all be zero.

In a class 1 call, if the base indicates that the EOF should be set to EOF - displacement, the server's current EOF will be determined and the EOF will be set relative to that; this could be different than the workstations assumption of the EOF if another workstation has modified the fork's EOF. This could also delete data that another workstation has written between the times when the current EOF was determined and the new EOF set.

This call will force any buffered data to be written to the server. The EOF will be set after this data is written.

GET_EOF ($19)
The fork's EOF will be determined from the server; this may not match the workstation's assumption of the EOF if another workstation has modified the fork's EOF. Note that another workstation could change the EOF after completion of this call, making the results inacurate.

This call will force any buffered data to be written to the server. The EOF will be determined after this data is written. This should avoid the problem of the returned EOF being less than the current mark.

GET_DIR_ENTRY ($1C)
Get_Dir_Entry is not supported for files. It will return the error $4E (access denied).

Folders enumerated by GET_DIR_ENTRY that have neither see files nor see folders will have the read bit in their access word cleared. Files, and folders with see files or see folders, will have the read bit set.

The access, filetype, auxtype, and option_list parameters are as the GET_FILE_INFO call. The FST will internally maintain the directory entry number (entry_num) to allow forward

and backward scanning of the directory. By default, several entries will be buffered for better performance (this can be disabled by using the FST Specific call "Buffer Control"). An end of directory error ($61) will be returned when an entry is requested that does not exist in the buffer (or buffering is disabled for the directory), and that entry cannot be read from the server.

Since AppleShare is a shared file system, entry_num may change for a file, even while the directory is being scanned because other users could add or delete files in the directory. Also, if the base and displacement fields are both zero, the total number of entries will be returned. Note that more or fewer entries may actually be returned if the directory is enumerated since other machines can create and delete files while you are enumerating the directory.

The best way to enumerate a directory is to simply open the directory and make successive Get_Dir_Entry calls with base and displacement both set to $0001. When you get an error $61 (end of directory), you are finished enumerating. You should remove duplicate entries from your list.

READ_BLOCK ($22)
This call will return an error $88 (network error) for AppleShare devices, in order to be compatible with System Disk 3.2. Remember, the preferred method for identifying a network volume is by doing a Volume call and seeing that the file_sys_id = $0D.

WRITE_BLOCK ($23)
This is an invalid operation for an AppleShare device. This call always return an error. The current error code is $4E (access denied). This is different from the $88 returned under 3.2, and may change in the future.

FORMAT ($24)
This is an invalid operation for an AppleShare device. This call always return an error. The current error code is $2B (write protected). This is different from the $88 returned under 3.2, and may change in the future.

ERASE_DISK ($25)
This is an invalid operation for an AppleShare device. This call always return an error. The current error code is $2B (write protected). This is different from the $88 returned under 3.2, and may change in the future.

GET_BOOT_VOL ($28)
If GS/OS is booted over AppleTalk, this command will return the name of the user volume on the server the user logged in to during booting. All system files should be present on this volume just like any other boot volume.

GET_FST_INFO ($2B)
The file_sys_id will be returned as $0D (AppleShare). The attribute parameter will be returned as $0000 (System Call Manager should not uppercase pathnames, do not clear high bits of pathname, this is a block FST, formatting not supported). The block_size parameter will be returned as 512; this value is only useful in determining the number of bytes used, free, and total on a volume (since these values are given in blocks).

FST_SPECIFIC ($33)

The FST_SPECIFIC call is used to make the Buffer Control, Byte Range Lock and Special Open Fork calls. The FST number must be $0D (AppleShare). A command of $0000 is invalid. Commands $000E through $FFFF are reserved.

If the command number is out of range, error $53 (invalid parameter) will be returned. Error $52 (unknown volume type) will be returned if a refnum for a file opened by another FST is used. Error $52 will also be returned if a pathname uses a device name for a device other than an AFP (AppleShare) driver. Error $45 (volume not found) will be returned if a pathname specifies a volume name that does not match any mounted AppleShare volume (even if a volume by that name exists for a different file system).

Buffer Control

Command $0001 is the Buffer Control command. It is followed by a word specifying the reference number of a file/directory whose buffering is to be enabled/disabled. The next word is optional. It specifies the buffer disable flags; if the high bit is set, then buffering is disabled for that file/directory. The default value of the buffer disable parameter is $0000 (turn on buffering). A file reference number of 0 is invalid.

For folders, the buffer size is 2048 bytes. When buffering is off, each Get_Dir_Entry will immediately cause an enumerate of one entry from the server. When a Get_Dir_Entry call is made with buffering on, the requested entry will be returned from the buffer if possible. Otherwise, the buffer will be filled with as many entries from the server as possible, including the requested entry; then the requested entry will be returned. The buffer is not pre-filled when the folder is opened. The number of entries kept in the buffer is variable and depends on the size of the long and short names of the files/folders.

For files, the buffer size is 512 bytes (the same as the block size reported by the FST). When buffering is off, every Read and Write call transfers data from/to the user's data buffer directly to/from the server. When buffering is on, and a Read or Write of 512 bytes or more is made, any unwritten data in the buffer is written and the Read/Write is made from/to the user's data buffer directly to/from the server.

When buffering is on and a Read or Write of less than 512 bytes is made, the block (512 bytes, with a starting offset that is a multiple of 512 bytes) containing the first byte to be read/written is read into the FCR's buffer; if the block was already in the buffer, no read is done; if a different block is in the buffer, any unwritten data is written and the new block is read into the buffer. The read/write then proceeds to the end of the buffer. If the read/write extends past the end of the buffer, any unwritten data is written and the next block is read into the buffer. The read/write then completes by reading/writing from/to the buffer.

Unbuffered reads with 0 or 1 newline characters are handled directly by the server (i.e. the read to the server requests the same number of bytes as the user requested). Unbuffered reads with 2 or more newline characters turn into reads of one character at a time from the server (until a newline is encountered or all bytes have been read or end of file reached); please note that this takes a LONG time, and you are probably better off not using 2 or more newline characters with buffering off.

Buffered reads with 1 or more newline characters become reads of 512 bytes at a time, on 512 byte boundaries (as if it were a read of less than 512 bytes). Each block is read into the FCR's buffer and then the bytes are copied to the user's data buffer one at a time (while being compared against all the newline characters). Buffered reads with no newline characters are as described above.

Byte Range Lock

Command $0002 is the Byte Range Lock command. It is followed by five required parameters (so the PCount field should be 7, 2 for FST # and Command, 5 for the parameters of Byte Range Lock). The first parameter is a word containing the reference number of the file to lock. The second parameter is the Lock Flag. If bit 15 is set, the range will be locked; if clear, it will be unlocked. If bit 14 is set, the offset is relative to the end of the file; if clear, the offset is relative to the start of the file. All other bits are reserved and should be set to 0. The next parameter is a long word containing the offset into the file (may be negative if relative to the end of the file). The next parameter is the length of the range to be locked. The last parameter is the actual start of the locked range (relative to the beginning of the file) as returned by the server.

Possible errors are: $4D (position out of range -- user already has some or all of range already locked, or unlocking a range not locked by that user), $4E (access denied -- some or all of range is locked by another user), $43 (invalid reference number), $53 (invalid parameter).

Special Open Fork

Command $0003 is the Special Open Fork command. It is followed by three required parameters and one optional parameter (so the PCount field should be 5 or 6: 2 for the FST# and Command, and 3 or 4 for the parameters of Special Open Fork). The first parameter is the reference number (ref_num) returned by GS/OS to the access path. Use this ref_num the same as you would a ref_num returned by an OPEN call. The second parameter is a pointer to a class 1 string representing the pathname of the file to be opened. The third parameter is the access mode giving the read/write permissions desired and to be denied to others as described below. The forth, and optional, parameter is the resource number: a value of $0000 will cause the data fork to be opened, a value of $0001 will cause the resource fork to be opened; a value of $0000 is assumed if the parameter is not given.

The access word is arranged as follows (if the bit is set, the condition is asserted):

Bit 0	Request Read Access
Bit 1	Request Write Access
Bit 2,3	Reserved
Bit 4	Deny Read to others
Bit 5	Deny Write to others
Bits 6..15	Reserved

Note: this parameter has the same meaning as in the ProDOS 8 Special Open Fork command.

Possible errors: same as for OPEN command. A deny mode conflict will result in an access denied error.

GetPrivileges

Command $0004 is the GetPrivileges command. It is followed by four parameters, the first two of which are required (so the minimum PCount is 4 and the maximum is 6). The first parameter is a pointer to a class 1 pathname of a directory whose access privileges are to be set or retrieved. The second parameter is a long where access rights for the directory will be returned. The third parameter is a pointer to a GS/OS output buffer where the owner's name will be stored. The fourth parameter is a poiner to a GS/OS output buffer where the group name will be stored.

The access rights field consists of four bytes: one each for user summary, world access, group access, and owner access. For each of these bytes, bit 0 is search access (see folders), bit 1 is read access (see files), and bit 2 is write access (make changes). The user

summary byte reflects the access that the current user has for that directory; if bit 7 is set, the current user is the owner of the directory.

If the folder is owned by the guest user (usually displayed as "<Any User>"), the owner name will be returned as a null string. If the folder has no group associated with it, the group name will be returned as a null string.

Possible errors include: $4B (bad storage type) if the pathname specifies a file instead of a folder.

SetPrivileges

Command $0005 is the SetPrivileges command. Its parameter list is the same as for the GetPrivileges command except that the access rights, owner name, and group name fields are input instead of output (since the values are being set, not retrieved). The owner name and group name point to structures similar to a GS/OS output buffer where the first word (normally a total buffer length) is ignored, the next word is the string length, and the rest of the buffer is the string itself. This structure definition allows you to do a GetPrivileges call, modify the data, and do a SetPrivileges call using the same owner name and group name pointers (the same way you can share the option_list parameter for Get_File_Info and Set_File_Info).

Setting the owner name to the null string assigns the folder to the guest user (usually known as "<Any User>"). The string "<Any User>" is not a valid user name (unless you have a registered user by that name). Setting the group name to the null string causes no group to be associated with the folder (and therefore the group's access rights are ignored).

Possible errors include: $4B (bad storage type) if the pathname specifies a file instead of a folder, $4E (access denied) if the user is not the current owner of the folder, $7E (unknown user) if the user name given is not the name of a registered user, and $7F (unknown group) if the group name given is not the name of a group.

User Info

Command $0006 is the User Info command. This command will return the user name and primary group name of a user. It has two required parameters and one optional parameter. The first parameter is the device number of a volume on the server whose user info is to be returned. The second parameter is a pointer to a GS/OS output buffer where the user name is returned. The third parameter (optional) is a pointer to a GS/OS output buffer where the user's primary group name is returned.

If the user is logged on as a guest, the user name will be returned as a null string. If the user has no primary group, then it will be returned as a null string.

Copy File

Command $0007 is the Copy File command. This command will cause a file on a server to be copied by the server. The copy may be between different volumes as long as both volumes are on the same server. This call has two required parameters. The first is a pointer to a class 1 string containing the source file's name. The second is a poiner to a class 1 string containing the destination file's name.

Possible errors include: $53 (invalid parameter) if either volume is not a server volume or if the volumes are not on the same server, $4A (version error) if the server does not support this call.

GetUserPath
Command $0008 is the GetUserPath command. It returns a pointer to a class 1 string containing the pathname of the user's folder on the user volume, using colons as separators and without a trailing colon. If there is no user volume mounted, or the user name could not be determined for some reason, a data unavailable error is returned ($60). This path is constructed on each call (unlike the FIUserPrefix call). The string's contents will not change until the next call to GetUserPath. DO NOT modify the string. The string is suitable for use as a parameter to a SetPrefix call.

OpenDesktop
Command $0009 is the OpenDesktop command. It takes a volume/path name and returns a desktop refnum (DTRefnum). A desktop refnum must be supplied for all other desktop database calls (currently, only for getting/setting file comments).

CloseDesktop
Command $000A is the CloseDesktop command. It takes a desktop refnum and volume/path name and frees all resources allocated when that refnum was opened.

GetComment
Command $000B is the GetComment command. It takes a DTRefnum and a pathname and returns a string (the comment associated with that file/folder). If no comment has been stored for that file/folder, then a null string will be returned for the comment.

SetComment
Command $000C is the SetComment command. It takes a DTRefnum, a pathname, and a string. If the string is non-null, then the comment for that pathname will be set to the given string. If the string is null, then the comment for that pathname will be removed. Note: if the comment string is longer than 199 characters, it will be truncated to 199 characters without an error.

GetSrvrName
Command $000D is the GetSrvrName command. It takes a pathname and returns the server name and zone name for that volume. If either of the server name or zone name buffer pointers are null ($0000 0000), that string will not be returned. If the server name or zone name are unknown, they will be returned as null strings.

General Implementation

When handling file system calls, the FST will itself create and send AFP packets to the server as opposed to trying to make the calls through PFI.

The only syntax checking that will be performed on pathnames is that the span (maximum length of a filename component) is less than or equal to 32 characters (the max for AFP); GS/OS itself will enforce the restriction against colons and nulls in a pathname component.

Normally, pathnames sent to the server will be relative to the root of the volume (i.e. the ancestor ID will be 2 = the volume directory). When a pathname is too long to fit in a packet, the FST will break it up into packet-size chunks by taking as many components from the start of the path as possible, finding its DirID, and repeat as needed until a DirID and partial path is obtained for accessing the file. This will cause more network traffic, but allow for long pathnames.

The FST is responsible for maintaining a File Control Record (FCR) for each open file. It will store at least the following: FCR refnum, pointer to the pathname, ID of owning FST, VCR ID of volume file is on, file level, pointer to newline list, newline count, newline mask, access byte.

The session/volume level information will be obtained from AppleShare device drivers. The .AFPn drivers maintain the relationship between an AppleShare volume and a Device Information Block (DIB). The FST maintains the Volume Control Record (VCR) for AppleShare volumes that are mounted.

If interrupts are disabled when the FST has to make an AppleTalk call (i.e. an SPCommand or SPWrite), an I/O Error (error code $27) will be returned instead of making the call. In most cases, this error will be propagated back to the user. Note that some calls may not require an AppleTalk call to be made (such as GetMark) and will complete correctly with interrupts disabled; some calls (such as Read and Write with small request counts, or GetDirEntry) may or may not complete with interrupts disabled (depending on the current mark, any data that is buffered, etc.). It is strongly encouraged that file system calls should not be made with interrupts disabled!

Appendix A – FST_SPECIFIC Calls

Buffer Control

word	Pcount (mininum = 3)		
word	FST# = $D		
word	Command = 1		
word	Reference #		
word	Buffer Disable flags (default = $0000)		
		Bit 15 set =	Disable buffering (every read/write goes to server,every GetDirEntry translated into a single FPEnumerate).
		Bits 0-14	Reserved

Byte Range Lock

word	PCount = 7			
word	FST# = $D			
word	Command = 2			
word	Reference #			
word	Lock Flag			
		Bit 15	set	Lock range
			clear	Unlock range
		Bit 14	set	Offset relative to EOF
			clear	Offset relative to start of file
long	Offset in File			
long	Length of Range			
long	Start of Range (returned)			

For the Lock Flag, the following constants can be combined:
Lock_Range = $8000
Relative_to_EOF = $4000

Special Open Fork

word	Pcount (minimum = 5)		
word	FST# = $D		
word	Command = 3		
word	Reference # (returned)		
long	Pointer to class 1 pathname		
word	Access mode		
		Bit 0	Request Read Access
		Bit 1	Request Write Access
		Bits 2,3	Reserved
		Bit 4	Deny Read to others
		Bit 5	Deny Write to others
		Bits 6..15	Reserved
word	Resource number (default =$0000)		

Get Privileges

word	PCount (min = 4)		
word	FST# = $D		
word	Command = 4		
long	Pointer to class 1 pathname		
long	Access Rights (returned)		
	byte User Summary		
		Bit 0	See Folders allowed
		Bit 1	See Files allowed
		Bit 2	Make Changes allowed
		Bits 3..6	Reserved
		Bit 7	Owner (set if you are folder owner)
	byte World		
		Bit 0	See Folders
		Bit 1	See Files
		Bit 2	Make Changes
		Bits 3..7	Reserved
	byte Group		
		Bit 0	See Folders
		Bit 1	See Files
		Bit 2	Make Changes
		Bits 3..7	Reserved
	byte Owner		
		Bit 0	See Folders
		Bit 1	See Files
		Bit 2	Make Changes
		Bits 3..7	Reserved
long	Pointer to GS/OS output buffer for Owner Name		
long	Pointer to GS/OS output buffer for Group Name		

Set Privileges

word PCount (min = 4)
word FST# = $D
word Command = 5
long Pointer to class 1 pathname
long Access Rights
- byte Reserved
- byte World
  - Bit 0 See Folders
  - Bit 1 See Files
  - Bit 2 Make Changes
  - Bits 3..7 Reserved
- byte Group
  - Bit 0 See Folders
  - Bit 1 See Files
  - Bit 2 Make Changes
  - Bits 3..7 Reserved
- byte Owner
  - Bit 0 See Folders
  - Bit 1 See Files
  - Bit 2 Make Changes
  - Bits 3..7 Reserved

long Pointer to buffer where Owner Name is stored (same format as a GS/OS output buffer, but the buffer length word is ignored).
long Pointer to buffer where Group Name is stored (same format as a GS/OS output buffer, but the buffer length word is ignored).

User Info

word PCount (min = 4)
word FST# = $D
word Command = 6
word Device number (of any volume on the desired server)
long Pointer to GS/OS output buffer for User Name
long Pointer to GS/OS output buffer for Primary Group Name

Copy File

word PCount (min = 4)
word FST# = $D
word Command = 7
long Pointer to class 1 string of source pathname
long Pointer to class 1 string of destination pathname

GetUserPath

word PCount (min = 3)
word FST# = $D
word Command = 8
long Pointer to class 1 string containing prefix (returned)

OpenDesktop

word PCount (min = 4)
word FST# = $D
word Command = 9
word Desktop refnum (returned)
long Pointer to class 1 string of path/volume name

CloseDesktop
word PCount (min = 4)
word FST# = $D
word Command = $A
word Desktop refnum
long Pointer to class 1 string of path/volume name

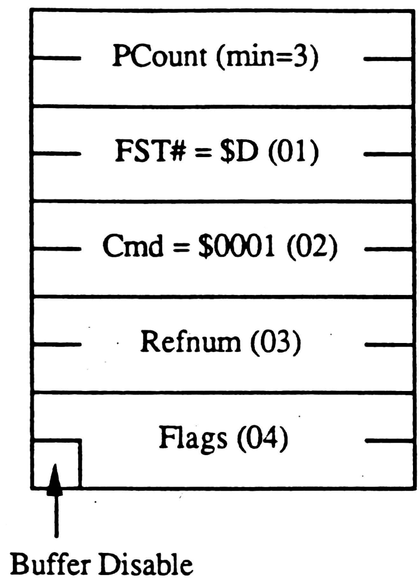
GetComment
word PCount (min = 5)
word FST# = $D
word Command = $B
word Desktop refnum
long Pointer to class 1 string of pathname
long Pointer to class 1 output buffer for comment

SetComment
word PCount (min = 4)
word FST# = $D
word Command = $C
word Desktop refnum
long Pointer to class 1 string of pathname
long Pointer to class 1 string of comment (default = null string)

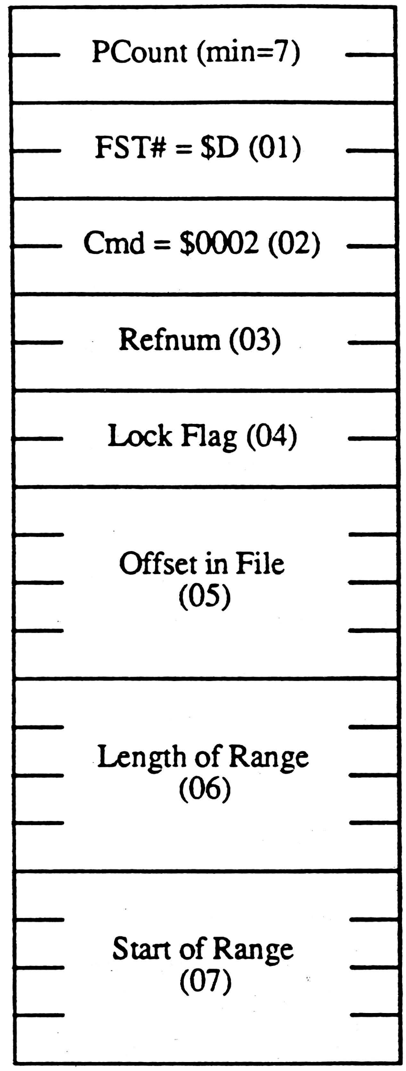
GetSrvrName
word PCount (min = 4)
word FST# = $D
word Command = $D
long Pointer to class 1 pathname
long Pointer to class 1 output buffer for server name
long Pointer to class 1 output buffer for zone name

Appendix B – Diagrams

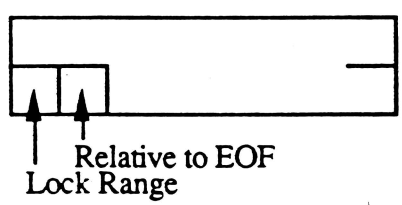
Buffer Control



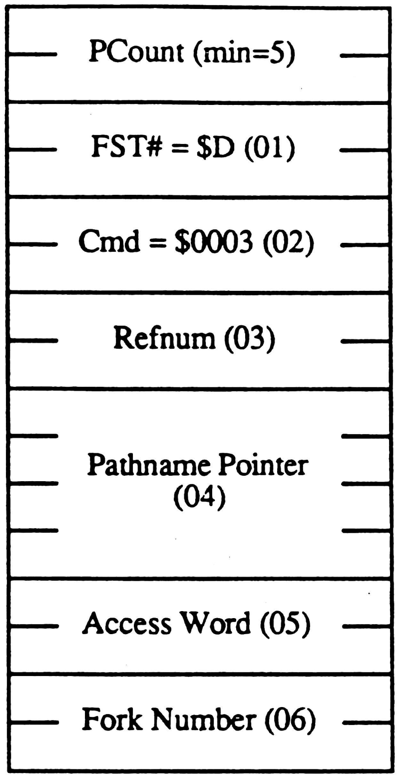
Byte Range Lock



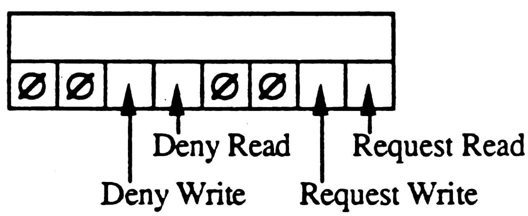
Lock Flag



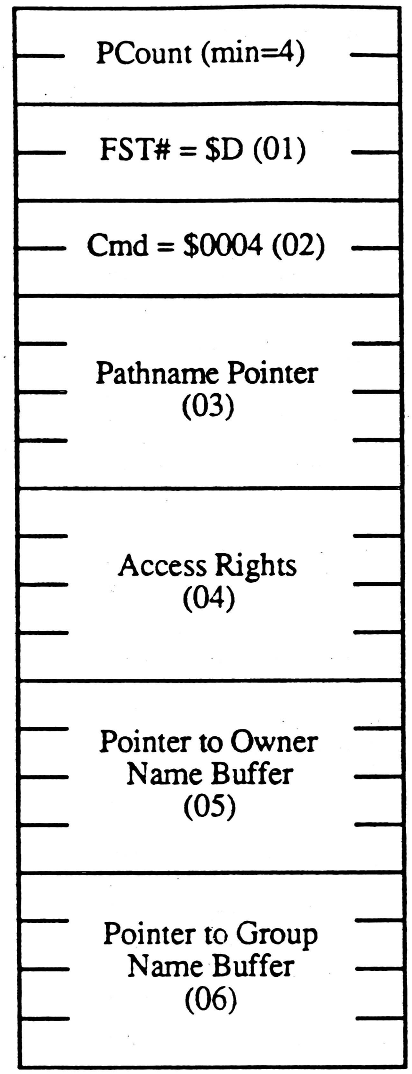
Special Open Fork

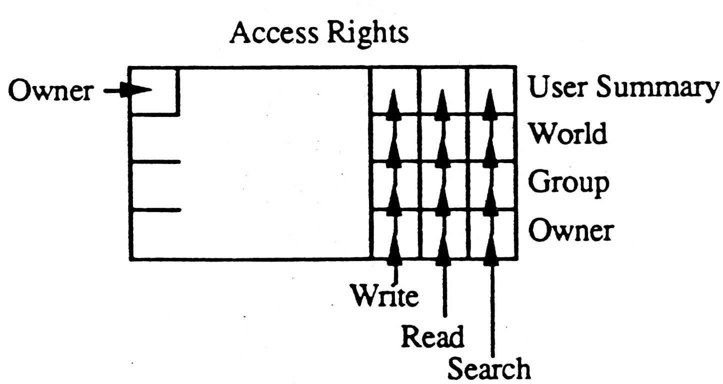


Access Word

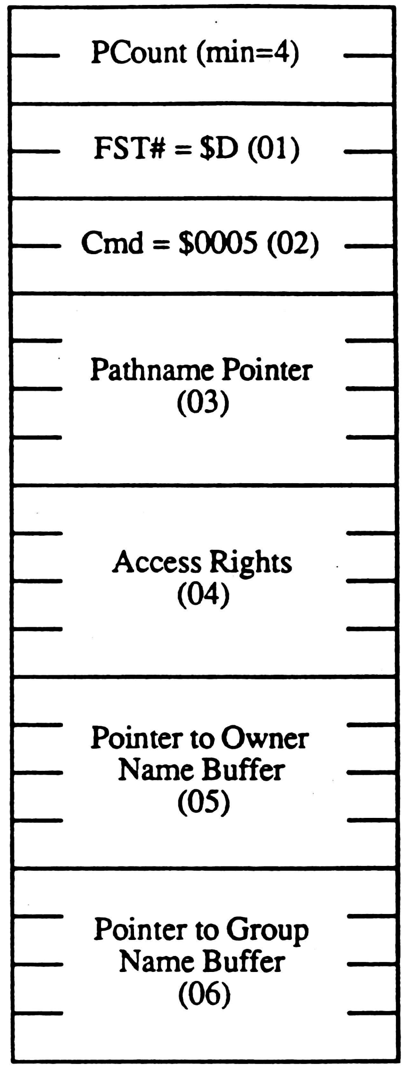


Get Privileges

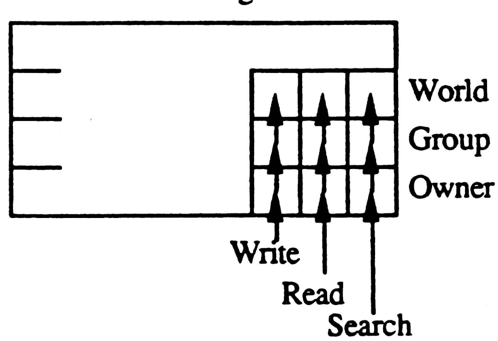




Set Privileges



Access Rights



User Info

PCount (min=4)
FST# = $D (01)
Cmd = $0006 (02)
Device Number (03)
Pointer to User Name Buffer (04)
Pointer to Pimary Group Name Buffer (05)

Copy File

PCount (min=4)
FST# = $D (01)
Cmd = $0007 (02)
Source Pathname Pointer (03)
Destination Pathname Pointer (04)

GetUserPath

PCount (min=3)
FST# = $D (01)
Cmd = $0008 (02)
Prefix Pointer (03)

OpenDesktop

PCount (min=4)
FST# = $D (01)
Cmd = $0009 (02)
DT Refnum (03)
Pointer to Volume Name (04)

CloseDesktop

PCount (min=4)
FST# = $D (01)
Cmd = $000A (02)
DT Refnum (03)
Pointer to Volume Name (04)

GetComment

PCount (min=5)
FST# = $D (01)
Cmd = $000B (02)
DT Refnum (03)
Pointer to Pathname (04)
Pointer to Comment Buffer (05)

SetComment

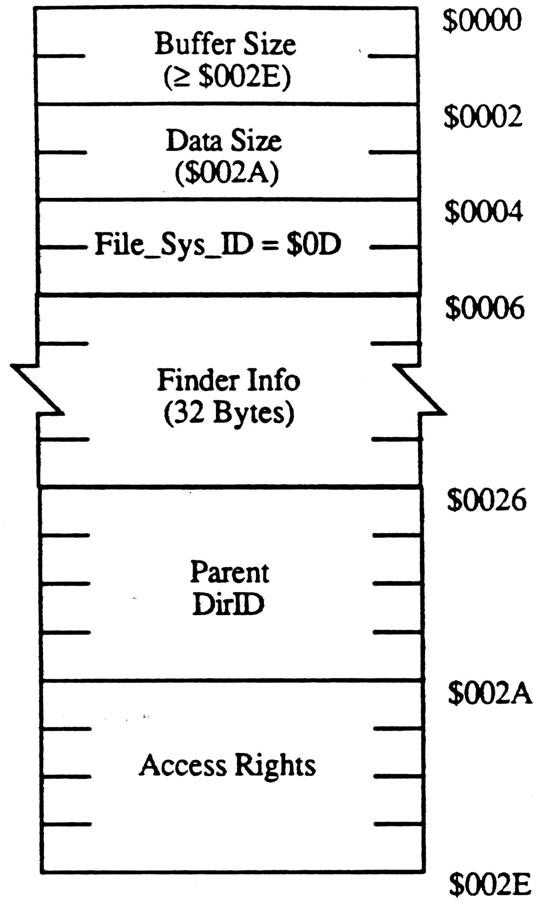
PCount (min=4)
FST# = $D (01)
Cmd = $000C (02)
DT Refnum (03)
Pointer to Pathname (04)
Pointer to Comment (05)

GetSrvrName

PCount (min=4)
FST# = $D (01)
Cmd = $000D (02)
Pointer to Pathname (03)
Pointer to Server Name Buffer (04)
Pointer to Zone Name Buffer (05)

Option_List

option_list



GS/OS AppleShare Booting
External ERS

Version 0.05

By Mark Day

Revision History

Date	Version	Who	Description
3-May-88	0.01	Mark Day	Describes algorithm and open issues.
3-June-88	0.02	Mark Day	Added support for booting multiple OS's. Removed suggestion for user-specific drivers and FSTs.
12-Sep-88	0.03	Mark Day	Removed support for multiple OS's since GS/OS supercedes ProDOS16.
20-Oct-88	0.04	Mark Day	Added description of user interface. Added description of Quick Logoff. Describes Aristotle patch for Quick Logoff.
| 14-Dec-88	0.05	Mark Day	Added custom setup files and DAs.

| Note: Changes from version 0.03 are denoted by a single bar in the left margin.

Introduction

This document describes the implementation of booting from an AppleShare file server. It is intended that AppleShare for GS/OS will be an upgrade for Toucan (AppleShare for ProDOS 16); therefore, GS/OS network booting will be an upgrade for ProDOS 16 network booting. This document also describes how the user will use the quick logoff feature.

Requirements

In order to boot GS/OS over the network, all servers in a zone should be updated with the GS/OS booting software. As with GS/OS in general, all machines that wish to boot GS/OS over the network must have version 01 ROMs or newer.

Downloading the Code

When AppleTalk is activated and the startup is set to AppleTalk, the AppleTalk ROMs will start the boot process. First it will look for an entity with type "Apple //gs"; the first object that responds will be used. It issues an ATP request with the machine type (1) in the first user byte and the block number of the image in the second and third user bytes (low order first, starting with block 0). Blocks are 512 bytes each and placed in memory starting at $800 in bank 0. If the first ATP user byte in the response from the server is non-zero then that is the last block in the image and it may be shorter than 512 bytes.

Because the retry counts and intervals in ROM prior to version 03 are too short when large numbers of machines are trying to boot, this first image will be one block or less in length to try to prevent timeout errors. This block will contain code that delays (about 5 seconds) to allow other machines booting up a chance to find the server before network traffic gets too heavy.

After the delay, a lookup for the operating system's image is performed. Then the first few blocks (about 2K) of this second image is loaded by the code in the first image using ATP requests. The block number and end-of-image flag work the same; the machine ID is

dependent on the operating system selected. After these few blocks have been read in, they receive control. This code is known as "Fizzy." Fizzy is responsible for displaying a user-friendly message and indicating progress while downloading the rest of the image. Data downloaded by Fizzy will be compressed to reduce network traffic and to improve booting speed.

Starting up the OS

The image contains patches and additions to the protocol stack through ASP, PFI, a logon program, and an FST stub. After protocol layers are intalled and initialized, the logon program runs allowing the user to log on to a file server. The FST stub containing the routines ReadInFile, GetBootName, and GetFSTName is left at $2000 and the file /Volume/System/Start.GS.OS is read in and executed at $6800. Start.GS.OS contains the GLoader and GQuit routines that will call the FST stub to load in the AppleShare FST from the System/FST directory on the boot volume. Once the FST is read in, the rest of the operating system will be loaded and executed.

At this time drivers, FSTs, setup files, DAs, etc. are loaded from folders in the System directory on the boot volume. Therefore, these files will be shared by all users who boot GS/OS from that server.

The AppleTalk drivers will find out from PFI which volumes were mounted and create Device Information Blocks for them so that the volumes the user selected during boot remain connected.

After the OS is loaded, it will first look for the file START in the System directory on the boot volume. The Start program will load any permanent or temporary init files and desk accessories found in the user's setup folder ("*/Users/*UserName*/Setup"), check for mail, open the user's ATINIT file, and launch their startup application. Note that the System/Start program will be run by GS/OS whenever an application quits and there is no program to quit to and there are no other programs waiting to be restarted (i.e. when ProDOS 16 would have displayed the "Start Next Program" menu).

GS/OS <—> ProDOS 8 Switching

ProDOS 8 will be loaded from the server on demand. PFI will have to be informed of any volumes mounted or unmounted while GS/OS was active so that ProDOS 8 will have an accurate view of the world (PFI will actually maintain session and volume information and will share this information with the AppleTalk drivers). After ProDOS 8 is loaded and initialized, PFI must be patched into the $BF00 vector.

The FST stub used during booting will be used by GLoader to reaload GS/OS when leaving ProDOS 8. When the drivers are reinitialized, they will have to find out from PFI if any volumes have been mounted or unmounted and update the Device Information Blocks appropriately. The FST stub should be much smaller than the part of ProDOS 16 kept around on the RAM disk for Toucan booting; therefore, GS/OS programs should have more free memory.

Trivia: AppleShare will be the only foreign file system that GS/OS can boot from and still support ProDOS 8.

Issues

The user interface will change from Toucan since a program separate from logon will do the mail check, setting the prefix, and launching the startup application. It has been suggested that the logon program display a nice message or picture and wait for the user to press a key before looking for AFP servers so that it will find servers that are responding when the user sits down at the machine (not a "stale" list of servers).

We need to be sure that the software is easy to install and upgrade. Note that many GS/OS files will have to be installed on the server before booting will work. Upgrading the protocols may be difficult since they are part of the image loaded during booting (unless they can be reloaded by the AppleTalk drivers). All other parts of GS/OS should be easy to upgrade by simply treating the server's user volume as any other boot volume and using the Apple IIGS Installer to update the system files.

User Interface (or How Do I Use This Thing, Anyway?)

To boot over the network, the first thing you need to do is set up the control panel properly. If you have version 01 ROMs, you need to set slot 7 to AppleTalk, and set the startup slot to 7. If you connect the drop box to the printer port, then slot 1 should be set to "Your Card" (slot 2 can be set to either "Your Card" or "Modem Port"). If you connect the drop box to the modem port, then slot 1 should be set to "Printer Port" and slot 2 should be set to "Your Card".

If you have version 03 ROMs, set slot 1 to "AppleTalk" if the drop box is connected to the printer port, or set slot 2 to "AppleTalk" if the drop box is connected to the modem port. Set "Startup:" to "AppleTalk".

As you power on (or reboot), you will see some dots displayed near the upper left corner of the screen; these dots are generated by the ROM and the first stage boot code to let you know that something is happening. At this point, the first stage of the boot code is being read from a server.

Shortly, the second stage boot code (known as "Fizzy") will have been loaded, and put up the screen shown in Figure 1.

The server name is displayed near the middle of the screen. A "spinner" (a line that rotates in 45° increments) is displayed between the "Starting up over the network" message and the server name; it indicates progress during the boot process by turning 45° as each block is read in. The thermometer at the bottom of the screen is filled in proportionately to the amount of boot information read in (it will be completely filled in as the last block is read in). A typical screen about 2/3 through the boot process is shown in Figure 2.

If the connection with the server is lost (i.e. a request times out), the server name, spinner, and the insides of the thermometer will be erased (reverting back to Figure 1) and there will be a lookup for another server. If a server is found, booting will start over; otherwise the screen will be cleared and control will return to the ROM to look for another server for the first stage boot code.

Once the AppleTalk protocols have been loaded and initialized, the Logon program will be run. If you have multiple zones or multiple servers in your zone, you will see the screen in Figure 3. If there are no bridges (and hence no zones), the "Current zone:..." string will not be displayed and "Change zones: Esc" will become "Cancel: Esc". If you press ESC at this point, an attempt will be made to find a bridge and let you choose from a list of zones (as in Figure 4). Once you have selected a zone or if there are no zones, you will be returned to the screen in Figure 3. Pressing ESC from the zone selection screen will pick your current zone and return to Figure 3.

To select from either the zone list or the server list, you may use the up and down arrow keys to move the highlighted bar up and down through the list. If there are more items below the bottom of the window, the word "More" is displayed along the bottom line of the window. Pressing Return will select the highlighted name from the list.

Once you have selected a file server (or if there are no zones and only one file server), you will be presented with the screen shown in Figure 5. You must choose whether you will log in as a guest ("<Any User>") or as a registered user. Pressing the up and down arrows will move through the choices. Pressing Return selects the highlighted choice. Pressing ESC returns you to Figure 3.

If you selected "Log on as a Registered User", you will be presented with the screen shown in Figure 6. You must enter your user name and password. If either is incorrect, a message will be displayed and you will be asked to try again.

Once you have successfully logged on to the server, you will be presented with a list of volumes on the server as shown in Figure 7. If there were no zones, only one server, and only one volume on the server, this list will not be displayed and the only volume will be automatically mounted. Volumes with a check mark next to them will be mounted when you press Return. Use the up and down arrow keys to move through the list of volumes; the left arrow key will remove the check mark next to the selected volume; the right arrow key will put a check mark next to the selected volume. Note that the user volume (the volume with the Users folder and the folders for all of the users) is automatically checked and cannot be unchecked. This volume will become your boot volume.

Once you have selected any additional volumes, the boot process will continue. Setup files, desk accessories, file system translators, drivers, etc. will be loaded from the user volume. Eventually, the startup application on the user volume will be run. If you have installed the network booting software normally, this will be the "*/System/Start" file and the process will continue as described below.

The startup application will first load any custom setup files and desk accessories found in your user folder ("*/Users/*YourName*/Setup"). Note that these are loaded in addition to the system-wide files loaded at boot time from the usual places in the System folder. Next, your mail folder ("*/Users/*YourName*/Mail") will be checked; if it is a non-empty folder, you will be told that you have mail waiting (see Figure 9). Your default printer will be set to the printer named in your ATINIT file (as set up in AppleShare Admin). Next, prefix 0 will be set to the prefix in the ATINIT file (set up in AppleShare Admin). Lastly, the user's startup application named in the ATINIT file (set up by AppleShare Admin) will be launched.

If the user's startup program quits, control will return to the AppleShare Startup program (described in the next section).

AppleShare Startup (and Quick Logoff)

The file '*/System/Start' is the AppleShare Startup program. If the user's startup application quits, control is returned to the AppleShare Startup program and the screen shown in figure 8 will be displayed. From here, you have four options: log off from all file servers, re-run the startup application, and reboot.

Selecting "Log off from file servers" will log you off from all file servers and then you (or another user) will be allowed to log on again (starting with figure 3 or 5 as appropriate). After logging on, the new user's startup application will be launched. Note that the operating system is not reloaded, and no custom desk accessories or setup files are loaded for the new user. This feature is known as Quick Logoff and is useful for quickly switching to a new user, such as in a classroom enviromment. Typically, a student would select this option at the end of a class and the student in the next class can log on without having to completely reboot.

Selecting "Re-run startup application" will check for mail and re-run your startup application as if you had just logged on (custom desk accessories and setup files will not be reloaded). Select this option if you accidently quit from your startup application and want to run it again.

Selecting "Reboot" will log you off from all file servers, eject all disks, and reboot the machine. This function is similar to the Restart option from the Finder's "Shut Down" command. This option should be used when you want to completely restart the computer, such as when you have loaded custom desk accessories or setup files and you don't want to have them installed for the next user.

The Aristotle Patch

Aristotle is Apple's classroom management software for the Apple II. When a user quits from Aristotle, it reboots the machine. Originally, this was done so that students would not have to run a separate Logoff program when they were done using the machine. This also means that Aristotle, as shipped, cannot make use of the quick logoff feature.

In order to allow Aristotle to take advantage of the quick logoff feature, we will include a patch program that will modify Aristotle to determine which machine it is running on before trying to reboot. If it is running on an Apple //e, it will reboot as ususal. On a IIGS, it will instead do a ProDOS 8 QUIT call to return control to the AppleShare Startup application.

The patch program will be a simple IIGS desktop application that will present the user with a Standard File dialog box. The user will select Aristotle's Menu.Display program. Once a file is selected, its size and a version string will be checked to be sure that the file is Aristotle. If the file is not Aristotle, an error message will be displayed and the user will be allowed to select another file. If the patch is already installed, the user will be so informed and allowed to select another file. If the patch has not been installed, the user will be asked to confirm whether they want to patch the file. The user may quit the program by clicking the cancel button on the dialog box.

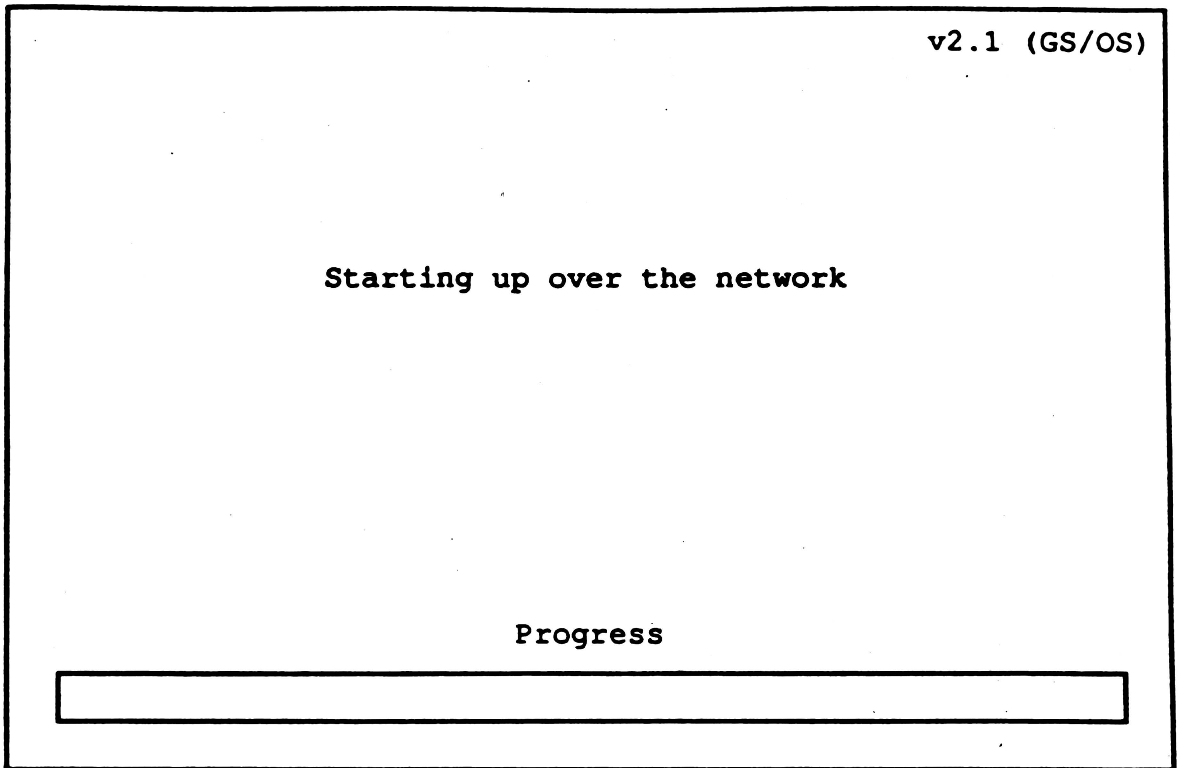


Figure 1

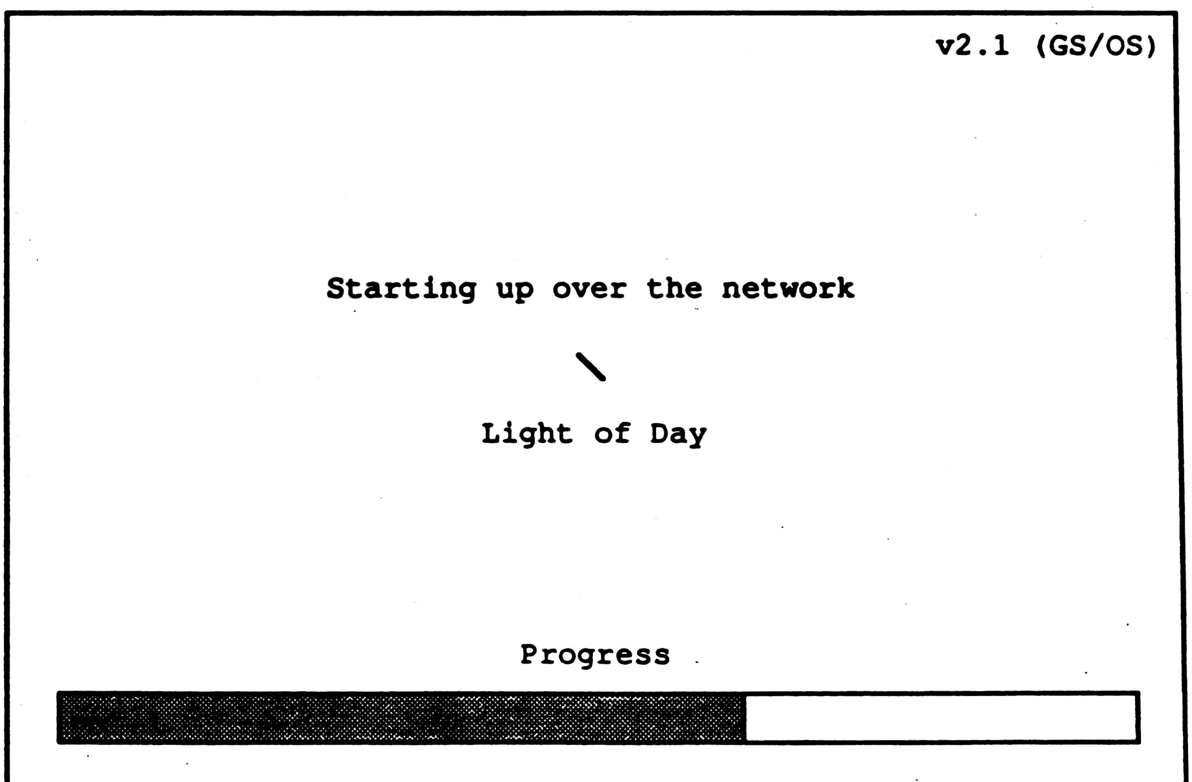


Figure 2

File Server Log On V1.1

Current zone: Nets-R-Us

Select a file server:

[illegible]
Light of Day
Aisle of View
We-B-Nets
Thunderhead
Ethersplat
Lots of Class

More

Select: ↑↓ Change zones: Esc Accept: ↵

Figure 3

File Server Log On V1.1

Current zone: Nets-R-Us

Select a new zone:

[illegible]
Net Com Doc
Gumby Zone
Metropolis
Engineering Support
The Breakfast Club
Twilight Zone

More

Select: ↑↓ Quit: Esc Accept: ↵

Figure 4

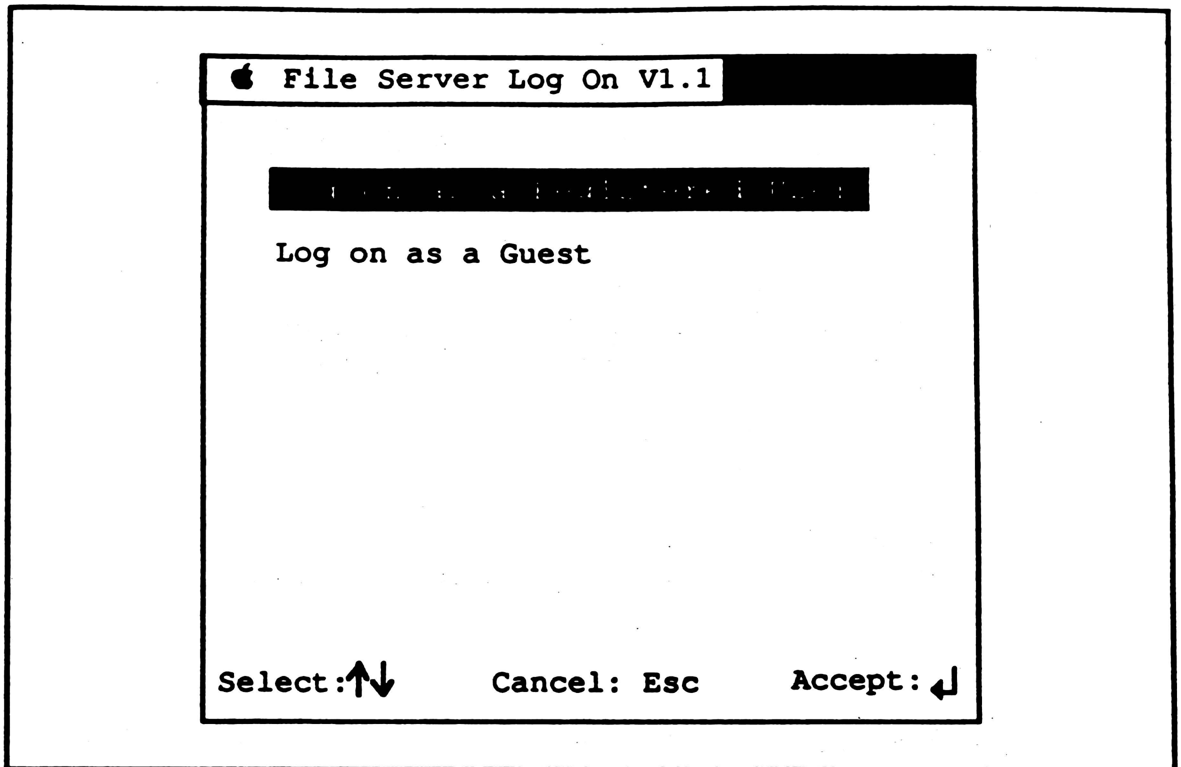


Figure 5

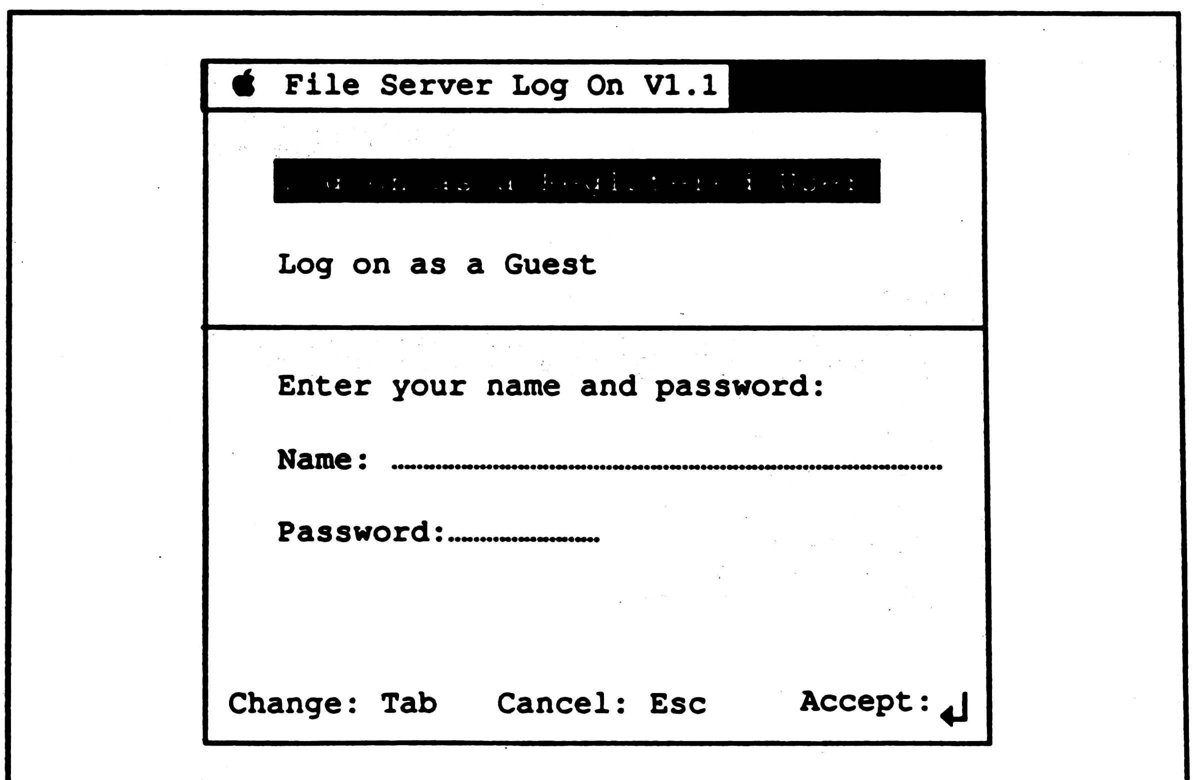


Figure 6

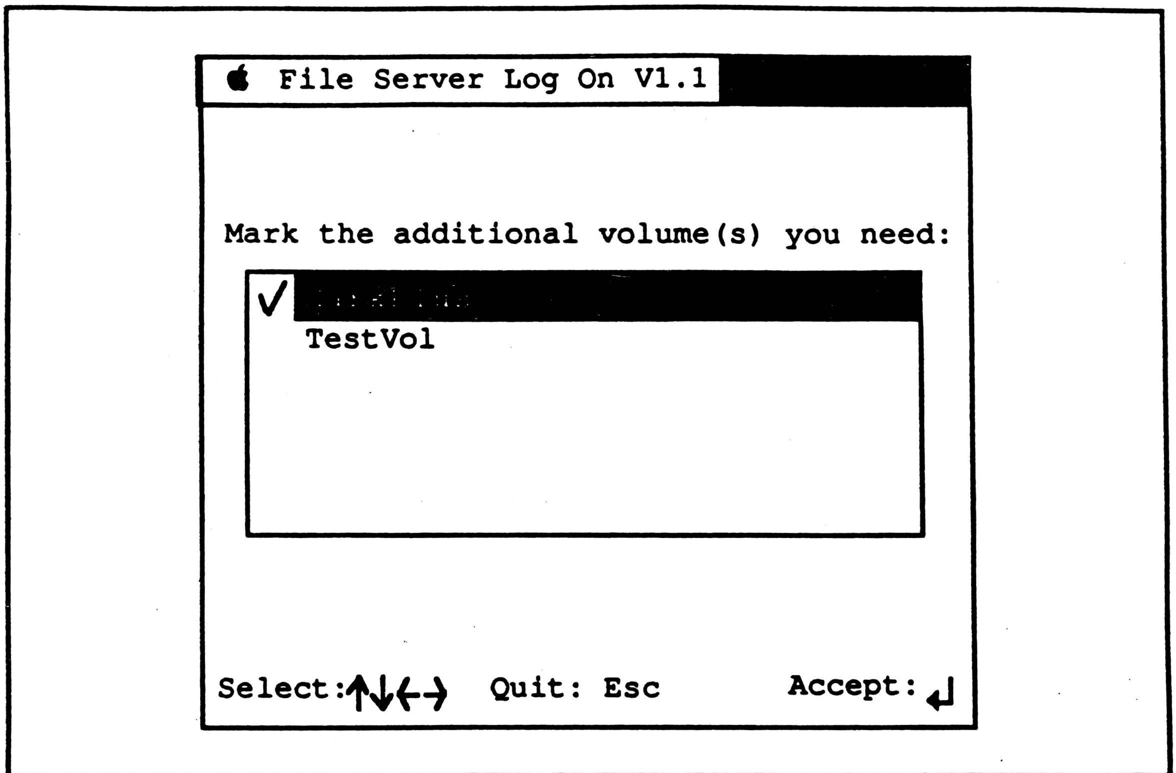


Figure 7

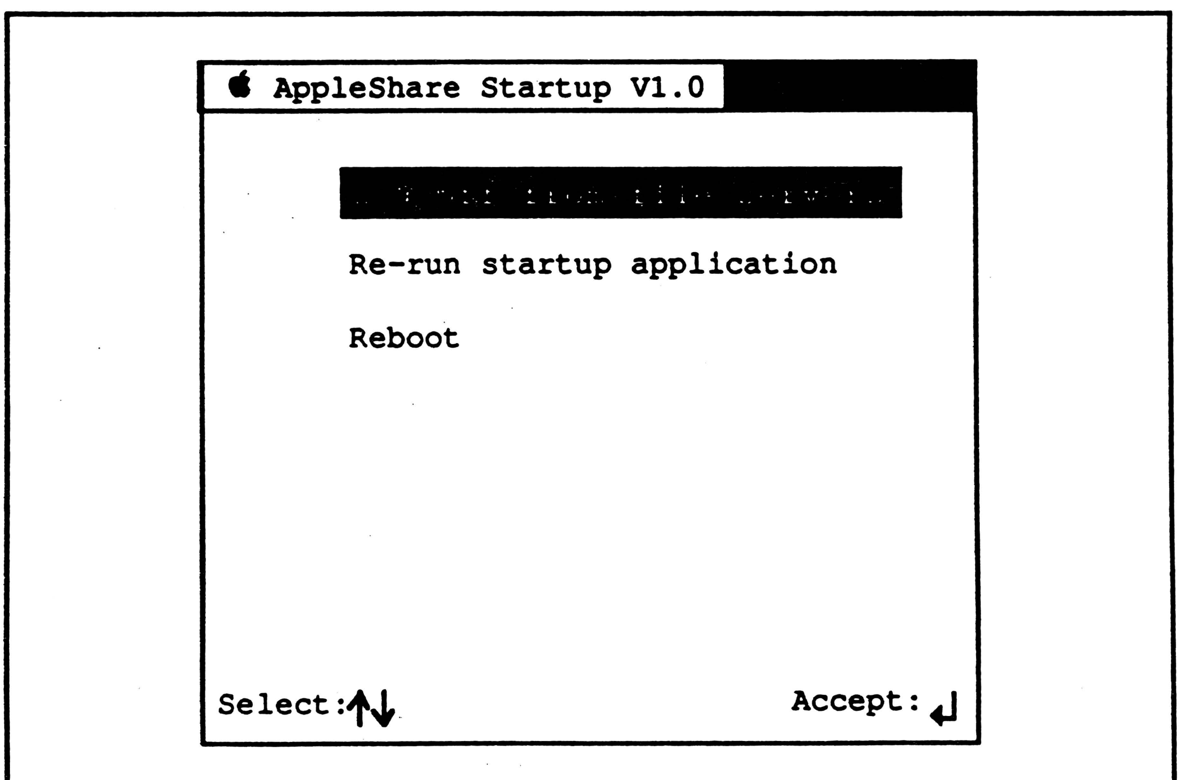


Figure 8

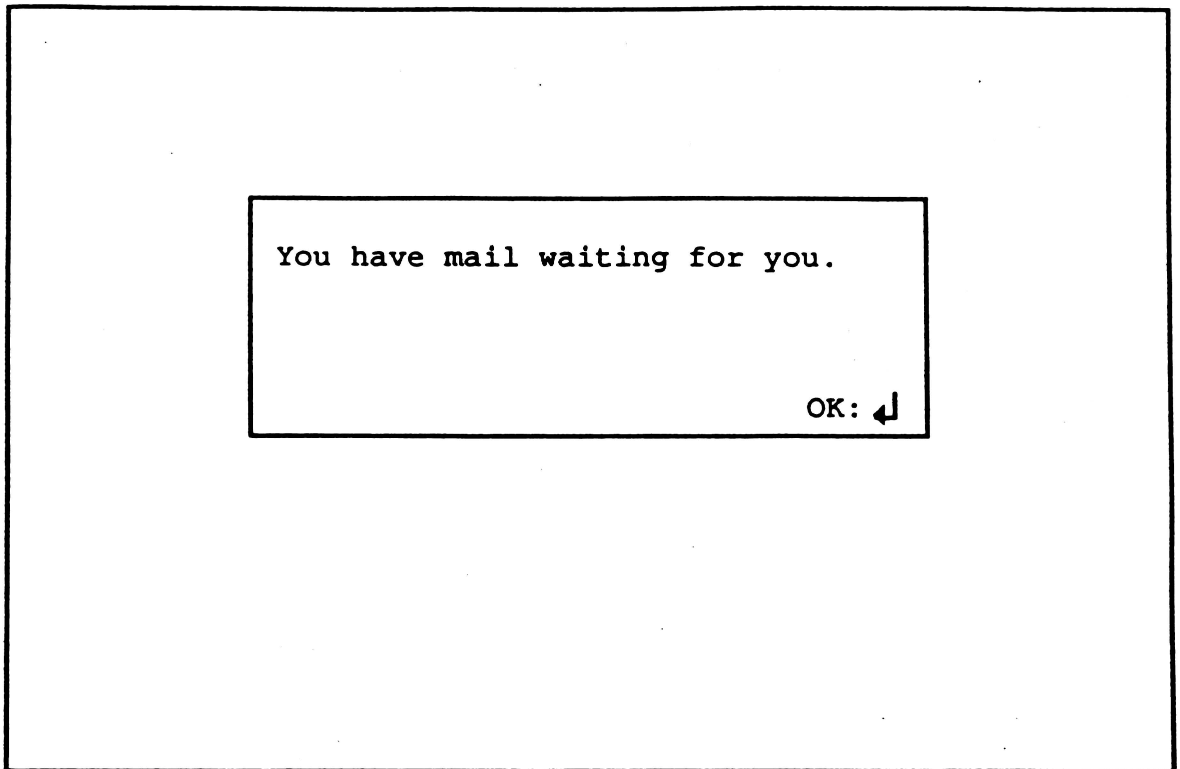


Figure 9

AppleTalk Driver ERS
(.RPM, .APPLETALK, & .AFPn)

EXTERNAL

Version 0.07

By Tim Harrington

Change History

Date	Version	Who	Description Of Changes
13-Sept-88	0.00	TGH	Initial Document Combined the .RPM, .APPLETALK, and .RPM drivers ERS into this one big ERS. Resolved some open issues. Took out unsupported call numbers.
13-Oct-88	0.01	TGH	Minor text changes.
13-Oct-88	0.02	TGH	Minor text changes. Defined GetPort control call in the .APPLETALK driver.
05-Dec-88	0.03	TGH	Minor text changes.
16-Dec-88	0.04	TGH	Changed status word for .APPLETALK and .AFPn drivers. Changed entity name definition for .RPM driver STATUS and CONTROL calls. Added Server and Zone name to .AFPn DIBs. Took out Open Issues chapter
3-March-89	0.05	TGH	Changed error codes for Read, Write, Status, and Control calls for the .AFPn and .APPLETALK drivers. Added descriptions to the standard STATUS and CONTROL to the .AFPn driver. Changed DIB values for all drivers. Added control call $8001 to .AFP driver.
27-March-89	0.06	TGH	Changed DIBS as to reflect that the drivers are now re-startable. Changed all references to Universe.
7-April-89	0.07	TGH	Added no-wait mode to .RPM driver. Changed error on WRITE call for the .AFP driver from 'access denied' to 'write protected'. Added detail to the .RPM status and control calls. Added Get and Set Eject Status calls to the .AFPn drivers.

About This Document

This document covers the design goals and calling mechanism for the AppleTalk driver. It assumes the reader already has knowledge of the following:

- AppleShare Programmer's Guide to the Apple IIGS document (formally the Toucan preliminary note - AppleTalk protocol layers for the Apple IIGS).
- GS/OS FST management.
- GS/OS device management.
- GS/OS generated drivers.

It is important that the reader know the difference between AppleTalk and AppleShare. AppleTalk is the entire network system and AppleShare is simply an AppleTalk service. AppleShare in this document refers to AppleTalk File Service.

This document is split up into three major sections. Each of these sections describes one of the three major components of the AppleTalk driver. The three components are drivers in themselves, but loosely connected to each other by being in the same driver file.

.RPM Driver

Background

In order to maintain compatibility with current ProDOS 8 (P8) applications, there needed to be a method to print to network printers with an already existing interface. Since most of the P8 applications already allow for the use of a Super Serial Card (SSC), the AppleTalk firmware in the AppleTalk slot was made to look and behave like an SSC. This allows current applications to print to network printers by making standard calls to the Pascal or BASIC entry points for the AppleTalk slot. The data to be printed will be collected by the firmware and sent to the Remote Print Manager (RPM). RPM will then send the data to a network printer in the proper format. RPM is capable of printing to either a network ImageWriter or a LaserWriter with the ImageWriter emulator installed.

Printing to RPM on the 4.0 GS/OS system disk is done through a generated driver that is created by GS/OS at boot time. GS/OS does this automatically for any slot that has valid firmware but no loaded driver. The *Universe* system disk will have a number of loaded drivers for the AppleTalk slot, therefore GS/OS will not generate a driver for RPM. This required a new RPM loaded driver to be written for the *Universe* system disk. This document will concentrate on the new .RPM driver that will provide network printing through the AppleTalk slot and RPM.

Overview

The .RPM driver is a loaded character driver which communicates with the AppleTalk Remote Print Manager (RPM). The purpose of this driver is to provide a means of communication between a GS/OS application and the older Pascal 1.1 entry points for the AppleTalk slot.

Instead of making calls directly to the AppleTalk slot, a GS/OS application can print to a network printer by making standard GS/OS WRITE calls to the .RPM driver. The driver will then use special GS/OS generated device management code to send the data to the AppleTalk slot. This is a much cleaner approach and GS/OS applications should make use of it. Although a GS/OS application can print to a network printer using the .RPM driver, every effort should be made to use the Print Manager. The .RPM driver is supplied for compatibility use only.

Device Driver Interface

The .RPM driver is part of the main ATALK driver in the */SYSTEM/DRIVERS/ subdirectory. Also included in this driver file will be the .APPLETALK and .AFPn drivers. Refer to the .AFPn and .APPLETALK driver sections for more information on those drivers.

Device Information Blocks

The main .RPM driver will have the following DIB settings:

Name	Size	Default	Description
LinkPointer	Long	0	Link pointer to next DIB
Entry Point	Long	Start of code	Points to .RPM driver's entry point
Characteristics	Word	$0B60	Read and Write access.
Block Count	Long	0	n/a for character device
Device Name	String[32]	'.RPM'	Default name of the .RPM driver
Slot #	Word	*	Slot number
Unit #	Word	*	Second unit number for this slot
Device Version	Word	$xxxx	Driver version x.xx experimental
Device ID #	Word	$001F	ID # (AppleTalk RPM driver)
Head Device	Word	0	Reserved
Link Device	Word	0	Reserved
Reserved1	Word	0	Reserved
Reserved2	Word	0	Reserved
DIB device #	Word	0	Device number gets put here

These additional DIB values are used as a GDIB. They follow the above standard DIB and can be accessed through standard Status and Control calls. The GDIB is needed in order to use the generated device slot emulation code.

Word	$0080	;Generated drvr_class (character)
Word	$0000	;Smartport type (none)
Word	$0000	;Smartport sub-type (none)
Word	$0000	;Open flag
Word	$0000	;Block size (none)
Word	$0000	;Hardware unit # (none)

The slot # and unit # are derived by calling the AppleTalk (SCC) supervisory driver at driver startup time.

Device Driver Calls

All calls followed by **OS only** can only be made by GS/OS itself. All calls followed by **FST only** can only be made by an FST.

DRIVER STARTUP (OS only) $0000

This call initializes the .RPM driver. This call will be issued by GS/OS at driver startup time. An application must NOT issue this call.

When this call is issued, the .RPM driver will call the AppleTalk (SCC) supervisory driver to determine its unit number and which slot AppleTalk is using and store those values in its DIB. If it cannot find the supervisory driver, the .RPM driver will not be inserted into the device list and AppleTalk will not be active.

DRIVER OPEN (FST only) $0001

This call is passed through to the generated device management code. This driver can only be opened once.

DRIVER READ $0002

This call is handled by the device driver directly and is not passed through to the generated device management code. This was necessary in order to provide no-wait read mode.

DRIVER WRITE $0003
DRIVER CLOSE (FST only) $0004

These calls are passed through to the generated device management code.

DRIVER STATUS $0005

This driver passes most of the standard status calls onto the generated device management code with the exception of the following calls:

$0002	Get Wait Status
$8001	Get RPM Parameters

All the status calls are described in more detail in the next section.

DRIVER CONTROL $0006

This driver passes most of the standard status calls onto the generated device management code with the exception of the following calls:

$0000	Reset Device
$0004	Set Wait Status
$8001	Get RPM Parameters

All the control calls are described in more detail in the next section.

DRIVER FLUSH (FST only) $0007
DRIVER SHUTDOWN (OS only) $0008

These calls are just passed through to the generated device management code.

Status And Control Calls

This section describes the status and control calls in more detail.

STATUS CALLS

Return Device Status $0000

This call returns the general device status word followed by a long word specifying the number of blocks supported by the device. This call is handled by the generated device management code.

Device Status Call Block

Offset	Field
0–1	Device Status
2–5	Block Count

Parameters

Device Status: word
<--
This is the device status word which is defined below.

0	1	2	3	4	5	6	7
x	0	0	0	1	0	0	0

8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0

Bit 0 is set if the device has been opened.

Block Count: long
<--
This value is not used by character devices and will always be equal to $00000000.

Return Configuration Parameters $0001

This call returns a byte count as the first word in the status list which indicates the length of the configuration parameter list. There is 0 bytes of configuration data.

Device Configuration Call Block

Offset	Field
0–1	Config. Length

Parameters

Config. Length word
<--
This is the length of the configuration parameter list in bytes. It will always be zero.

This call returns the current wait mode.

Parameters

Wait Mode word

<--

$0000 = Wait mode
$8000 = No wait mode

Get Wait Status Call Block

Offset	Field
0	Wait Mode
1	

Get Format Options $0003
Get Partition Map $0004

These calls do not pertain to character drivers and will always return with no error and the transfer count = 0. These calls are handled by the generated device management code.

This call returns the current parameters used by RPM.

Get RPM Parameters Call Block

Offset	Field
0–1	Result
2–5	Entity Name Pointer
6	Flags
7–8	Flush Interval
9–A	Timeout Interval
B–C	Number of Buffers

Result: word
<--

The result from the AppleTalk call will be placed here. See the Toucan documentation for a complete list of valid error codes.

Entity Name Pointer: pointer
-->

This is a 4 byte pointer to a standard AppleTalk entity name. An entity name is a character string consisting of three Pascal strings: object, type, and zone concatenated together. The first two fields can be up to 32 characters plus 1 length byte. The third field can be up to 33 characters plus 1 length byte. This means that the supplied buffer must be 100 bytes in length.

Flags: byte
<--

This field will hold the flag byte for RPM.

7	6	5	4	3	2	1	0
x	0	x	0	0	0	0	0

Bit 7 is set when the specified printer is on the network. This bit is always set on the Apple IIgs.
Bit 5 is set if the printer is to be sent the postscript emulation string.

Flush Interval: word
<--

This is a word value specifying the number of 1/4 seconds RPM should wait for new characters before flushing the out-going buffer.

Timeout Interval: word
<--

This is a word value specifying the number of 1/4 seconds RPM should wait for new characters before timing out and ending the print job.

Number Of Buffers word
<--

This is a word value specifying the number of 512 byte buffers RPM should use for storing out-going information. The more buffers set, the faster the writing will be.

CONTROL CALLS

Reset Device	$0000

This call will reset the specified device to it's default settings. The only setting that is affected by the reset is the wait/no-wait read mode. After a reset the read mode will be set to wait. This call will always return with no error and the transfer count = 0.

Format	$0001
Eject Media	$0002
Set Configuration Parameters	$0003

These calls do not pertain to character drivers or do not apply and will always return with no error and the transfer count = 0. These calls are handled by the generated device management code.

Set Wait Status	$0004

This call sets the current wait mode.

Parameters

Wait Mode word -->

$0000 = Wait mode
$8000 = No wait mode

Set Wait Status Call Block

Offset	
0	Wait Mode
1	

Set Format Options	$0005
Assign Partition Owner	$0006
Arm Signal	$0007
Disarm Signal	$0008
Set Partition Map	$0009

These calls do not pertain to character drivers or do not apply and will always return with no error and the transfer count = 0. These calls are handled by the generated device management code.

Set RPM Parameters	$8001

This call sets the given parameters to be used by RPM.

Result: word <--

The result from the AppleTalk call will be placed here. See the Toucan documentation for a complete list of valid error codes.

Entity Name Pointer: pointer -->

This is a 4 byte pointer to a standard AppleTalk entity name. An entity name is a character string consisting of three Pascal strings: object, type, and zone

concatenated together. The first two fields can be up to 32 characters plus 1 length byte. The third field can be up to 33 characters plus 1 length byte.

Flags: byte

-->

The field will hold the flag byte for RPM.

7	6	5	4	3	2	1	0
x	0	x	0	0	0	0	0

Bit 7 is set when the specified printer is on the network. This bit should always be set on an Apple IIgs.
Bit 5 is set if the printer is to be sent the postscript emulation string.

Set RPM Parameters Call Block

Offset	Field
0–1	Result
2–5	Entity Name Pointer
6	Flags
7–8	Flush Interval
9–A	Timeout Interval
B–C	Number of Buffers

Flush Interval: word

-->

This is a word value specifying the number of 1/4 seconds RPM should wait for new characters before flushing the out-going buffer. This value must be less that the Timeout Interval. Normally this field should be set to 2 (1/2 second).

Timeout Interval: word

-->

This is a word value specifying the number of 1/4 seconds RPM should wait for new characters before timing out and ending the print job. This value must be greater that the Flush Interval. Normally this field should be set to 120 (30 seconds).

Number Of Buffers word

-->

This is a word value specifying the number of 512 byte buffers RPM should use for storing out-going information. The more buffers set, the faster the writing will be. Normally this field should be set to 20 (10K of buffer space).

.APPLETALK Driver

Background

In the past, the only way for an application or driver to determine general AppleTalk parameters was to look at specific memory locations. Because we never did, and never will, publish those memory locations, it is currently quite impossible. With this new .APPLETALK driver, an application or FST can determine these general AppleTalk values through a legal means.

Overview

The .APPLETALK driver is a loaded character driver which provides a common interface for determining general AppleTalk variables. The driver provides the following parameters:

- **AppleTalk Presence.**
  An Application can make the assumption that if this driver is present in the system, the AppleTalk protocols (LAP through PFI) are also present.

- **AppleTalk Port.**
  This returns which SCC channel the Link Access Protocol (LAP) is using as the AppleTalk port.

Interaction With The Protocol Layers

This driver uses and provides no direct communication with the AppleTalk protocol layers. An application must still make AppleTalk protocol calls through the AppleTalk dispatch vector at $E11014. The reasoning behind this is that AppleTalk uses an asynchronous calling scheme and GS/OS cannot handle this properly.

Device Driver Interface

The .APPLETALK driver is part of the main ATALK driver in the */SYSTEM/DRIVERS/ subdirectory. Also included in this driver file will be the .RPM, .AFP and .AFPn drivers. Refer to the .AFP and .RPM sections for more information on those drivers.

Device Information Block

The .APPLETALK driver will have the following DIB settings:

Name	Size	Default	Description
LinkPointer	Long	0	Link pointer to next DIB
Entry Point	Long	Start of code	Points to driver's entry point
Characteristics	Word	$0F00	Loaded, Char Dev, Not-speed dep.
Block Count	Long	0	n/a for character device
Device Name	String[32]	'.APPLETALK'	Default name of the driver
Slot #	Word	*	Slot number
Unit #	Word	*	Unit number for this slot
Device Version	Word	$xxxx	Driver version x.xx experimental
Device ID #	Word	$001D	ID # (.APPLETALK driver ID #)
Head Device	Word	0	Reserved
Link Device	Word	0	Reserved
Reserved1	Word	0	Reserved
Reserved2	Word	0	Reserved
DIB device #	Word	0	Device number gets put here

The slot # and unit # are derived by calling the AppleTalk (SCC) supervisory driver at driver startup time.

Device Driver Calls

All calls followed by **OS only** can only be made by GS/OS itself. All calls followed by **FST only** can only be made by an FST.

DRIVER STARTUP (OS only) $0000

This call initializes the .APPLETALK driver. This call will be issued by GS/OS at driver startup time. An application or FST must NOT issue this call.

When this call is issued, the .APPLETALK driver will call the AppleTalk (SCC) supervisory driver to determine the unit number for this driver and which slot AppleTalk is using and store those values in its DIB. If it cannot find the supervisory driver, the .APPLETALK driver will not be inserted into the device list and AppleTalk will not be active.

DRIVER OPEN (FST only) $0001

This call will allow the driver to be opened multiple times. This call is only supported for compatibility reasons.

DRIVER READ $0002
DRIVER WRITE $0003

The device dispatcher will never dispatch these calls to the .APPLETALK driver and will return error $4E, Invalid Access, instead.

DRIVER CLOSE (FST only) $0004

This call will allow the driver to be closed only if it had been previously opened. This call is only supported for compatibility reasons.

DRIVER STATUS $0005

This call is used to request status information from the driver. This driver supports all the standard calls plus the device specific call listed below. Error codes depend on the call being made. The device dispatcher intercepts many non-character device calls and returns the appropriate errors.

$8001	GetPort

This call is described in more detail in the next section.

DRIVER CONTROL $0006

This call is used to send control information to the driver. This driver supports all the standard calls. Error codes depend on the call being made. The device dispatcher intercepts many non-character device calls and returns the appropriate errors. The exception to this is the arm and disarm signal calls. These calls will always return a 'bad control call' error.

DRIVER FLUSH (FST only) $0007
DRIVER SHUTDOWN (OS only) $0008

These calls have no effect and always returns 'No Error'.

Status And Control Calls

This section describes the device specific status call in more detail. The standard status and control calls behave as described in the GS/OS reference manuals.

STATUS CALLS

GetPort $8001

This call returns the port number that AppleTalk is currently using.

Parameters

Port Number: word
<--

This is the port number which AppleTalk is currently using.

1 = Port 1 (Printer Port)
2 = Port 2 (Modem Port)

GetPort Status Block

0–1	Port Number

.AFPn Drivers

Background

AppleTalk file service for the Apple IIGS was first introduced with the Toucan Workstation disk. This was achieved through a near impossible piece of code called the ProDOS Filing Interface (PFI). PFI provided a means for both ProDOS 8 and ProDOS 16 to communicate to AFP file servers. Then just when we thought we were safe, the native mode operating system, ProDOS 16, is to be replaced by GS/OS. With the release of this new operating system, AppleTalk file service support will have to be re-written. This includes enhancements to the protocol layers, a number of new drivers, and an AppleTalk FST. This document will concentrate on the driver that will provide the FST with a means of logically connecting AFP volumes to GS/OS.

Overview

The .AFPn is actually 14 drivers in one. It has 14 DIBs for 14 different AppleShare volumes but uses the same core routines for all 14 DIBs.

The .AFPn driver is a loaded block driver which communicate with the AppleTalk ProDOS Filing Interface (PFI). The purpose of this driver is to handle the following conditions:

- **Maintaining and updating all AppleShare DIBs.**
  Particularly the loss of a connection due to either logging off or the loss of the session with a server. It treats this event as a disk switch. The only way for the driver to again hold valid data is if a new volume is mounted and assigned to this driver.

- **Notify user of lost connection and server shutdown.**
  This will be accomplished through beeping the internal speaker and flashing the boarder colors as it was for system disk 3.2.

- **Provide session information.**
  This includes the session reference number, volume ID, volume name, server name, and zone name associated with a particular .AFPn driver.

- **Handle volume eject.**
  This is done by unmounting volume and logging off the server if unmounting the last volume associated with that server.

Interaction With PFI

Because the ProDOS Filing Interface (PFI) provides AppleTalk file service for ProDOS 8, it must still be resident under GS/OS. Instead of duplicating much of the code that is already in PFI, the .AFPn driver will rely heavily on PFI in determining information about AFP volumes. The .AFPn driver will rely on PFI for the following:

- Notified when a new volume is mounted or unmounted.

- What volumes are already mounted during an OS switch between ProDOS 8 and GS/OS.

- AFP volume names and their session reference numbers and volume ID numbers.

- Notified when an attention message comes in from a server.

Device Driver Interface

The .AFPn drivers are part of the main ATALK driver in the */SYSTEM/DRIVERS/ subdirectory. Also included in this driver file will be the .RPM and .APPLETALK drivers. Refer to the .RPM and .APPLETALK sections for more information on those drivers.

Device Information Blocks

The .AFPn driver will have the following DIB settings for each of its DIBs:

Name	Size	Default	Description
LinkPointer	Long	0	Link pointer to next DIB
Entry Point	Long	Start of code	Points to .AFPn driver's entry point
Characteristics	Word	$0BE4	Device characteristics
Block Count	Long	$007FFFFF	n/a for AppleTalk volume
Device Name	String[32]	'.AFPn'	Default name of the .AFPn driver
Slot #	Word	*	Slot number
Unit #	Word	*	Second unit number for this slot
Device Version	Word	$xxxx	Driver version x.xx experimental
Device ID #	Word	$001E	ID # (AppleTalk AppleShare driver)
Head Device	Word	0	Reserved
Link Device	Word	0	Reserved
Reserved1	Word	0	Reserved
Reserved2	Word	0	Reserved
DIB device #	Word	0	Device number gets put here

The second part of the .AFPn drivers DIB is device dependent data. This data is $65 (101) bytes in length. It's used by the driver to identify and hold information on each .AFPn driver.

Device Number	Word	0	;.AFPn unit number (0..13)
Session Number	Byte	0	;Session reference number
Reserved	Byte	0	;Slot/Driver under P8
Volume Name	String[28]	0	;Volume name
Volume ID #	Word	0	;Volume ID number
Status Word	Word	0	;Device status word
Server Name	String[32]	0	;Server name
Zone Name	String[33]	0	;Zone name

The slot # and unit # are derived by calling the AppleTalk (SCC) supervisory driver at driver startup time. The device name, volume name, server name, and zone name are all

Pascal strings. String[32] means that the string is 32 bytes long INCLUDING the length byte.

Device Driver Calls

All calls followed by **OS only** can only be made by GS/OS itself. All calls followed by **FST only** can only be made by an FST.

DRIVER STARTUP (OS only) $0000

This call initializes the .AFPn driver. This call will be issued by GS/OS at driver startup time. An application or FST must NOT issue this call.

When this call is issued, the .AFPn driver will call the AppleTalk (SCC) supervisory driver to determine the unit number for this driver and which slot AppleTalk is using and store those values in its DIB. If it cannot find the supervisory driver, the .AFPn driver will not be inserted into the device list and AppleTalk file service will not be available.

Just before the .AFPn exits from this routine, it asks PFI if there any AppleShare volumes are already mounted. This will happen at startup and when switching from P8 to GS/OS. If any AppleShare volumes are mounted, the .AFPn driver will assign a new DIB for every AppleShare volume found.

DRIVER OPEN (FST only) $0001

This call has no effect and always returns 'No Error'.

DRIVER READ $0002

In order to maintain compatibility with system disk 3.2, this call will normally return 'AppleTalk Error' ($88). The exceptions to this is 'Disk Offline', 'Disk Switched', and 'Bad Request Count' errors.

DRIVER WRITE $0003

This call has no effect and always returns the error 'Write Protected'.

DRIVER CLOSE (FST only) $0004

This call has no effect and always returns 'No Error'.

DRIVER STATUS $0005

This call is used to request status information from the driver. This driver supports all the standard control calls plus one additional device specific call. Error codes depend on the call being made.

Following is the only valid device specific control code:

$8002	Get Eject Status

All the status calls are described in more detail in the next section.

DRIVER CONTROL $0006

This call is used to send control information to the driver. This driver supports all the standard control calls plus two additional device specific calls. Error codes depend on the call being made.

Following are the only valid device specific control codes:

$8001	Display Messages
$8002	Set Eject Status

All the control calls are described in more detail in the next section.

DRIVER FLUSH (FST only) $0007

This call has no effect and always returns 'No Error'.

DRIVER SHUTDOWN (OS only) $0008

This call will log the user off all file servers and return with 'No Error'.

Status And Control Calls

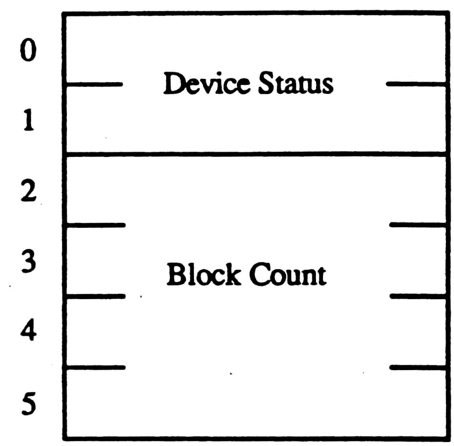
This section describes all the status and control calls in more detail.

STATUS CALLS

Return Device Status $0000

This call returns the general device status word followed by a long word specifying the number of blocks supported by the device.

Device Status Call Block



Parameters

Device Status: word
<--

This is the device status word which is defined below.

0	1	2	3	4	5	6	7
x	0	0	0	x	0	0	0

8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	1

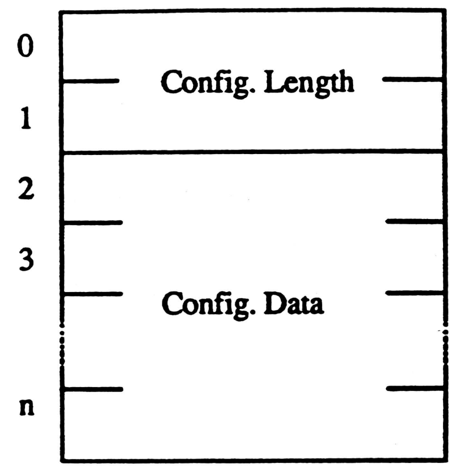
Bit 4 is set if volume is on-line.
Bit 0 is set if the volume has gone off-line.

Block Count: long
<--

This value will always be equal to $007FFFFF for an AppleShare volume.

This call returns a byte count as the first word in the status list which indicates the length of the configuration parameter list. The configuration parameters will be placed into the status list contiguous to the byte count. There is 101 bytes of configuration data. The request count must be no more than 103; the two extra bytes being the Config. Length field.

Device Configuration Call Block



Parameters

Config. Length word
<--

This is the length of the configuration parameter list in bytes.

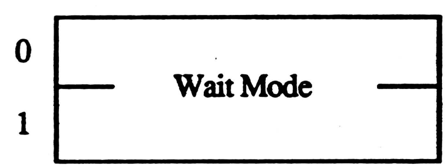
Config. Data block
<--

This is the actual configuration data. The standard DIB settings as defined earlier in this document will be returned in Config. Data.

Get Wait Status $0002

This call returns the current wait mode.

Get Wait Status Call Block



Parameters

Wait Mode word
<--

$0000 = Wait mode
$8000 = No wait mode

This driver will always be in Wait mode.

Get Format Options $0003
Get Partition Map $0004

These calls do not pertain to this driver because it cannot be formatted. These calls will always return with no error and the transfer count = 0.

Get Eject Status $8002

This call returns the current eject status.

Parameters

Eject Status word
<--

$0000 = Volume may be ejected
$8000 = Volume may not be ejected

Get Eject Status Call Block

Offset	Field
0	Eject Status
1	

The default is $0000 (Volume may be ejected.) This value will not be re-set to zero when a RESET control call is made or when switching back from ProDOS 8 to GS/OS..

CONTROL CALLS

Reset Device $0000

This call will reset the specified device to it's default settings. This call is supported for compatibility only and actually does nothing. This call will always return with no error and the transfer count = 0.

Format $0001

This call does not pertain to this driver because it cannot be formatted. This call will always return with no error and the transfer count = 0.

Eject Media $0002

This call will unmount the volume and log the user off the file server if it was the last volume connected to that server. This call will always return with no error and the transfer count = 0.

Set Configuration Parameters $0003

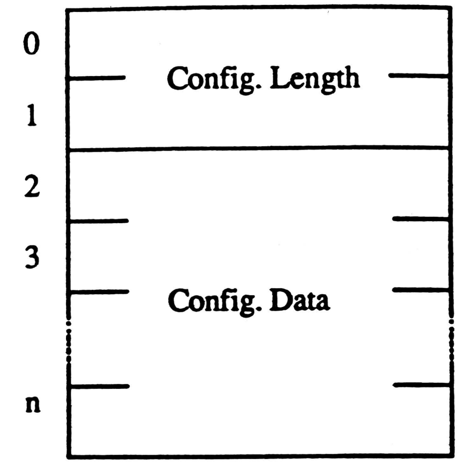
This call would normally set the current parameter list of the specified device to the given parameter list. Because the beginning of the AFP DIB configuration data contains critical information, the first 101 bytes are ignored and never set by this call. The first word of the status buffer must be a byte count which indicates the length of the configuration parameter list. The configuration parameters should be contiguous to the byte count.

Config. Length word
-->

This is the length of the configuration parameter list in bytes.

Parameters

Device Configuration Call Block



Config. Data block
-->

This is the actual configuration data. Because the Config. Data contains critical information, the first 101 bytes are ignored and never set by this call.

Set Wait Status $0004

This call sets the current wait mode. A read from this driver will always result in error $88 so this call does nothing and always returns with no error and the transfer count = 0.

Set Format Options $0005
Assign Partition Owner $0006

These calls do not pertain to this driver because it cannot be formatted. These calls will always return with no error and the transfer count = 0.

Arm Signal $0007
Disarm Signal $0008

The .AFPn drivers do not support any signaling so this call will always return with a 'Bad Control/Status Code' error and the transfer count = 0.

Set Partition Map $0009

This call does not pertain to this driver because it cannot be formatted. This call will always return with no error and the transfer count = 0.

Display Messages $8001

This call will display any server messages that are pending. This call is for internal use. There is no need for a program to make this call.

Get Eject Status $8002

This call sets the current eject status.

Parameters

Eject Status word -->

$0000 = Volume may be ejected
$8000 = Volume may not be ejected

Set Eject Status Call Block

Offset	
0	Eject Status
1	

This value will not be re-set to zero when a RESET control call is made or when switching back from ProDOS 8 to GS/OS.

GS/OS: Be AppleShare Aware

External ERS

Version 0.01

By Mark Day

Revision History

Date	Version	Who	Description
6-Mar-89	0.01	Mark Day	First writing. Covers multi-launch applications, the Open call, interrupts, multi-user applications.

Introduction

An "AppleShare aware" program is a program that can be successfully run from an AppleShare file server. Such a program should be able to load and save files on a file server, and be fully functional. It should be able to handle error conditions in a reasonable manner (such as putting up a dialog box instead of crashing the machine), and the user should be able to quit from the program and return to a calling program (instead of having to reboot or power off the machine).

This document describes some steps you can take as a developer to help make your programs AppleShare aware. It also describes some things you can do to make your programs even more usable in an AppleShare environment (such as being multi-launch), and how to take advantage of some AppleShare-specific features.

Multi-launch applications

A multi-launch application is one that can be launched (executed) by more than one computer at a time. Multi-launch applications are particularly important for the Apple II family since most schools use Apple II's and it is common for an entire class to use the same application at the same time. Teachers are much more likely to use a multi-launch application on a file server than to distribute individual disks for each student.

ProDOS has traditionally been a single user, single computer operating system and file system. With the addition of AppleShare support to GS/OS, many computers (and many types of computers) can share the same files (on the file server) at the same time. It is not hard to make a program multi-launch; it just takes some thinking and care about how you use files.

The first thing to remember about multi-launch applications is that one copy of the application will be shared by several computers. The system loader will take care of opening and loading the file in a safe manner such that several computers can load the application at the same time. As the programmer, you must remember that you should not write to the application files (to save configuration information, for example) just like you shouldn't write in books borrowed from a library -- other people have to use it, too.

GS/OS 4.1 has a new feature called the "@" prefix. It is a system prefix defined when your application is launched. If the application was launched from an AppleShare volume, it will be set to the name of the user's folder on the file server. If the application was launched from a non-AppleShare volume, it will be set to the name of the folder containing the application. If you use the "@" prefix as part of the pathname for saving configuration information, it will automatically go in a safe place, separate for each user. For example, if your program was called "Fred", you might use the pathname "@:Fred.Config" for storing preferences and configuration data.

Sharing open files

The class 1 version of the Open call lets you supply a parameter indicating the access you require to the open file. You can specify read, write, read and write, or "as permitted". If you request read permission (request_access=1), it will also deny others the ability to write to the file (so they can't change the data you are reading). If you request write or read and write (request_access =2 or 3, respectively), it will deny others the ability to open the file at

all (so they cannot read data as you are changing it, and so they cannot overwrite your changes to the data). Realize that "as permitted" (request_access=0) will first try to open the file for read and write (meaning no other computer can open it); if that fails, it will try read-only; if that fails, it will try write-only. Note that there is no way of knowing what access you have to the file, and you may not have read/write access. If your program opens files, think about how it uses the contents of the files, and open them in an appropriate manner.

For example, an adventure game might want to load a map of rooms in a dungeon. In this example, the program really only needs to read the contents of the file, and not modify the file. Since all you need to do is read the file, you should open the file read-only (request_access=1). If you do this, and several computers run the program at the same time, they will all be able to open the dungeon file successfully (since a read-only open allows others to open the file read-only). If you use request_access=0, or don't even supply the field (it is optional), only the first computer will be able to open the file; the rest will get an error when trying to open the file (access denied, $4E).

As a second example, consider a word processing program. It would want to read from the file so that it can be displayed or printed. It would also want to write to the file so that it can be edited and save the changes. In this case, the program would open the file with request_access=3 (read and write). Don't assume that request_access=0 will give you read and write access; other users who have opened the file, or access privilege settings may restrict your access. Also, the file should be kept open the entire time the file is being edited. If you don't, another computer could open the file for editing after you have closed it. Then, the edited version that is written last will stay, and all other versions will be overwritten.

As a third example, consider a file copying program (like the Finder). It would open the source file read-only (so that other computers can copy it or use it). It would open the destination file write-only (request_access=2) since it only needs to write to the file, and no other computer should be allowed to read or write to the copy while it is being written. Note that opening the destination for read and write could cause the open to fail if access privileges to the file prevent read access (such as if the file is in a "drop box").

The class 0 version of the Open call is compatible with the ProDOS16 Open call. Since it did not provide a mechanism to tell the operating system what access was needed to the file, it allows files to be opened in a manner that is not completely safe in order that several computers could open the same file at the same time (the first computer to open the file could potentially change it as other computers are trying to read from it). The class 1 Open call is safe, and allows you to specify the access that you require to the file.

All authors are strongly encouraged to use the class 1 version of the Open call and to use a non-zero value for the request_access field. This way, files can be shared if possible, and if the open succeeds, you will know that you have the access to the file that you need.

Interrupts

AppleTalk needs to have interrupts enabled to function correctly. When interrupts are off, packets cannot be received from or sent to other computers. This will cause network services to stop functioning. One particularly visible aspect of this problem is losing a connection with a file server. It only takes four consecutive missed packets for the workstation to assume the server has shut down or has become unreachable.

Do not leave interrupts disabled any longer than absolutely necessary. Beware that if interrupts are disabled inside a loop, that the effect is multiplied by the number of iterations. Leaving interrupts disabled for just a few microseconds could cause a packet to be missed. Obviously, there are some times when interrupts must be disabled, such as in a critical timing loop for a disk driver.

Interrupts must be on for an incoming packet to be received. Therefore, repeatedly turning interrupts on and off can be just as bad as leaving them off the entire time. For example, if a section of code has interrupts disabled 80% of the time and enabled 20% of the time, you will miss approximately 80% of all incoming packets.

Remember, interrupt handlers (like heartbeat tasks) execute with interrupts off. Keep their run time as short as possible (such as setting a flag for a foreground task to check).

Do not make operating system calls with interrupts disabled. These calls could potentially take long periods of time to complete (for example, a large file read). AppleShare calls will not be able to complete with interrupts disabled.

Multi-user applications

A multi-user application is an application that lets several users access and possibly change some common data at the same time. A multi-user application is usually multi-launch. A typical example is a database program that lets several users view and edit records at the same time. In this case, the read/write protections are applied to individual records instead of the entire file. Doing this requires using some commands specific to AppleShare.

First, you would use the FST_Specific call SpecialOpenFork to open the file (fork). With this call you not only provide the access you want to the file, but the access you will allow others to the file. For example, a database file might be opened for read/write, deny nothing. This way, all users can open the file and read and write to it at the same time. (buffering off).

To prevent one workstation from writing to the file and corrupting information being read or written by another workstation, you use the FST_Specific call ByteRangeLock. It takes an open file refnum, some flags, an offset into the file, and a length. The (length) number of bytes starting at the given offset can be locked or unlocked. When a range of bytes is locked, no other workstation can read or write those bytes; in fact, the same workstation using a different refnum cannot access those bytes. Note that you can lock a range past the EOF of the file, which is necessary when extending the size of the file.

For example, you might want to add a new record to a database. First, you would lock the header of the file and read it in to determine where to place the new record. Then you would lock the range where the new record will be located. Next, update the header to indicate the new record has been allocated, write out the header, and unlock it. Now, write the new record to the range you have locked, and unlock the range.

Remember that you should have locked any range of bytes that you are reading or writing, and that you should re-read a range of bytes if you have unlocked and locked it again. Note that buffering is disabled by default for the SpecialOpenFork call to prevent inconsistencies between the buffer's and the file's contents (with the normal Open call, this

is not a problem since no other workstation is allowed access that could cause such an inconsistency).

Additions to the AppleShare Programmer's Guide for the Apple IIGS

(formerly: Toucan Preliminary Note)

Delta ERS - EXTERNAL
May 2, 1988
v0.03

by
Tim Harrington
Network Systems Development

Change History

Date	Version	Who	Description Of Changes
2/22/88	0.01	TGH	Added change history section. Added section on changes to PMSetPrinter. Added the FIGetSVersion call.
3/3/88	0.02	TGH	Added AFP version word to Login2.
5/1/88	0.03	TGH	Added the Introduction and Asynchronous vrs. Synchronous paragraphs. Added details to the FIHooks call.

Introduction

This document explains the changes and additions made to the AppleTalk® protocol stack for the Apple IIGS™ computer and System Disk 5.0. It is assumed that the reader has read and has access to the following documents:

- *AppleShare Programmer's Guide to the Apple IIGS*
- *Inside AppleTalk*

Asynchronous vrs. Synchronous

The programmer should never make a synchronous only call with the async flag set (bit 7 = 1). Although some synchronous only calls can be made with the async flag set, the results can be unpredictable. In most cases, the call will complete with no detectable side effects, but others will hang or crash.

New Miscellaneous Calls

CancelTimer ($45)

The CancelTimer call is used to cancel an asynchronous InstallTimer call before it completes. It uses the identical parameter list as the corresponding InstallTimer that is being removed. The parameter structure for the RemoveTimer call list listed here.

Position	Name	Size	Value
$00	Async Flag	Byte	$00 (Synchronous only)
$01	Command	Byte	$45
$02	Result Code	Word	<---
$04	Reserved	12 Bytes	x (the rest of the InstallTimer parameter list)

The Async Flag and Command bytes must be changed in the original parameter list used for the InstallTimer call. If the timer routine has not been installed or had completed, a "No Timer Installed" error ($0103) is returned. If the timer is successfully canceled, the completion routine will receive a "Timer Canceled" error ($0106). This is important as a successful result for the CancelTimer call will return error $0106 instead of "No Error" ($0000).

The CancelTimer call returns these result codes, as well as the result codes for all system calls.

Result	Description
$0103	No Timer Installed
$0106	Timer Canceled

New Calls to the Name Binding Protocol (NBP)

NBPKill ($46)

The NBPKill call is used to cancel an asynchronous NBP call before it completes. The parameter structure for the NBPKill call list listed here.

Position	Name	Size	Value	
$00	Async Flag	Byte	$00	(Synchronous only)
$01	Command	Byte	$46	
$02	Result Code	Word	<---	
$04	ParamBlockPointer	Long	--->	

ParamBlockPointer must point to the beginning of the parameter block that is currently being used by the asynchronous call that is to be canceled.

The NBPKill call returns these result codes, as well as the result codes for all system calls.

Result	Description
$040A	ParamBlock Not Found

New Calls to the Remote Print Manager (RPM) Interface

PMSetPrinter ($28)

This call hasn't changed accept for the fact that if the Timeout Interval is set to zero (0), then the session will never time out and must be stopped via the new PMCloseSession call.

PMCloseSession ($47)

The PMCloseSession call is used to close any outstanding RPM session. The parameter structure for the PMCloseSession call list listed here.

Position	Name	Size	Value	
$00	Async Flag	Byte	$00	(Synchronous only)
$01	Command	Byte	$47	
$02	Result Code	Word	<---	

The PMCloseSession call never returns an error.

New Calls to the ProDOS Filing Interface (PFI)

PFI has been modified to correctly set and get invisibility status on files. This includes both regular files and directories. Bit 2 in the ProDOS Access field now specifies whether or not the file or directory is invisible.

FIHooks ($37)

The FIHooks call is used for changing the default event notification routine. If the login program passes the default attention routine (null) to PFI, the default hooks will be called. These default hooks can be either set or returned through this call. The parameter structure for the FIHooks call is listed here.

Position	Name	Size	Value	
$00	Async Flag	Byte	$00	(Synchronous only)
$01	Command	Byte	$37	
$02	Result Code	Word	<---	
$04	Flag Byte	Byte	--->	
$05	MountVector	Long	<-->	
$09	UnmountVector	Long	<-->	
$0D	AttentionVector	Long	<-->	

The Flag Byte field specifies the OS type and whether the hooks are to be set or returned, as shown in Table 1.

Table 1. Bit Settings for the Hook Flag Field

Bit Number	Setting	Description
7	Set (1)	ProDOS 8 active.
	Clear (0)	GS/OS active.
6	Set (1)	The hooks will be set.
	Clear (0)	The hooks will be returned.
5 - 0	Clear (0)	Must be zero.

> Note: If bit 6 is clear, hooks to be returned, then bit 7 is ignored and the OS type will not be changed.

The MountVector field is a pointer to the routine that will be called whenever PFI adds a new volume to its internal tables.

The UnmountVector field is a pointer to the routine that will be called whenever PFI removes a volume from its internal tables. The MountVector and UnmountVector will be called in the following environment:

• = Undefined

ENTRY:	Called via 'JSL'	
	A Reg =	Undefined
	X Reg =	Low word of parameter block pointer
	Y Reg =	High word of parameter block pointer
	D Reg =	PFI direct page
	B Reg =	PFI data bank
	P Reg =	N V M X D I Z C E
		• • 0 0 0 • • • 0

The parameter block contains the following data:

Byte	Session reference number
Byte	P8 Unit #
PString[28]	Volume name
Word	Volume ID
PString[32]	Server name
PString[33]	Zone name

EXIT:	Return via 'RTL'	
	A Reg =	Undefined
	X Reg =	Undefined
	Y Reg =	Undefined
	D Reg =	PFI direct page
	B Reg =	PFI data bank
	P Reg =	N V M X D I Z C E
		• • 0 0 0 • • 0 0

The AttentionVector field is a pointer to the routine that will be called whenever PFI receives a standard attention event for one of the mounted volumes.The AttentionVector will be called in the same environment as the mount and unmount vectors with the following parameter block:

Byte	Session reference number
Byte	Type of attention
Word	Attention data
PString[32]	Server name
PString[33]	Zone name

The result codes returned for the FIHooks call are the same as those common to all general system calls.

FILogin2 ($38)

The FILogin2 call is used to log in to a server. This call work primarily like the FILogin call. The exception is that there are two additional parameters at the end of the FILogin call structure. The parameter structure for the FILogin2 call is listed here.

Position	Name	Size	Value
$00	Async Flag	Byte	$00 (Synchronous only)

$01	Command	Byte	$38
$02	Result Code	Word	<---
$04	SLS Network Number	<Word>	--->
$06	SLS Node Number	Byte	--->
$07	SLS Socket Number	Byte	--->
$08	Command Buffer Length	Word	--->
$0A	Command Buffer Pointer	Long	--->
$0E	Reply Buffer Length	Word	--->
$10	Reply Buffer Pointer	Long	--->
$14	Session Reference #	Byte	<---
$15	Attn Routine	Long	--->
$19	Server Name Pointer	Long	--->
$1D	Zone Name Pointer	Long	--->
$21	AFP Version Number	Word	--->

The Command Buffer must be in AFP format for the FILogin2 call, with the first 2 bytes reserved for the AFP Command Number. When the call completes, the Reply Buffer contains the reply, if any, in AFP format. The Session Reference # field will return the ASP Session Reference Number. If the call completes with the Login Continue Error, the caller must complete the log-in process with the server by using the FILoginCont call. As far as PFI is concerned, the session has been established, unless the call completes with an error other than Login Continue.

The Server Name Pointer and Zone Name Pointer must point to a valid Pascal String (length byte followed by name). The AFP Version word must be in the following format:

"AFPVersion 1.1"	=	0101 (hexadecimal)
"AFPVersion 2.0"	=	0200 (hexadecimal)

The high byte is the major version number and the low byte is the minor version number.

The Server Name, Zone Name, and AFP Version fields are NOT used by PFI to login to the server. These fields are required for the ListSessions2 and FIGetSVersion calls. It is up to the programmer making the Login2 call to verify that these parameters are correct.

The FILogin2 call returns these result codes, as well as the result codes for all system calls.

Result	Description
$0A01	Too many sessions
$0A02	Unable to open session
$0A03	No response from server
$0A04	Login continue
$0A13	Already logged in to server
$0A15	User not authorized
$0A16	Parameter error
$0A17	Server going down
$0A18	Bad UAM
$0A19	Bad version number

FIListSessions2 ($39)

The FIListSessions2 call is used to retrieve a list of current sessions being maintained through PFI and any volumes mounted for those sessions. This call work primarily like the FILogin call. The exception is that there are two additional parameters returned for every session. The parameter structure for the FIListSessions2 call is listed here.

Position	Name	Size	Value	
$00	Async Flag	Byte	$00	(Synchronous only)
$01	Command	Byte	$39	
$02	Result Code	Word	<---	
$04	Buffer Length	Word	--->	
$06	Buffer Pointer	Long	--->	
$0A	Entries Returned	Byte	<---	

The list is placed into the specified buffer. If the buffer is not large enough, the buffer will retain the maximum possible number of current sessions and then return as error. The format of the buffer is as follows:

Position	Name	Size	Value
$00	Session Reference #	Byte	<---
$01	Slot/Drive	Byte	<---
$02	Volume Name	28 Bytes	<---
$1E	Volume ID	<Word>	<---
$20	Server Name	32 Bytes	<---
$40	Zone Name	33 Bytes	<---

The FIListSessions2 call returns these result codes, as well as the result codes for all system calls.

Result	Description
$0A0B	Buffer too small

FIGetSVersion ($3A)

The FIGetSVersion call is used to determine what version of AFP was used to login to a particular server. The parameter structure for the FIGetSVersion call is listed here.

Position	Name	Size	Value	
$00	Async Flag	Byte	$00	(Synchronous only)
$01	Command	Byte	$3A	
$02	Result Code	Word	<---	
$04	Session Number	Byte	--->	
$05	AFP Version Number	Word	<---	

The AFP Version word will be in the following format:

0101 (hexadecimal)	=	"AFPVersion 1.1"
0200 (hexadecimal)	=	"AFPVersion 2.0"

The high byte is the major version number and the low byte is the minor version number.

The FIGetSVersion call returns these result codes, as well as the result codes for all system calls.

Result	Description
\$0A06	Invalid session reference number

Cache Manager Delta

version 0.02

By
Rob Turner

Revision History:

version 0.01 03/02/89	first version.
version 0.02 03/15/89	first version.

Cache Manager Enhancement:

(1) System Disk 5.0 contains a new cache manager that adds an additional feature. The new feature, referred to as AutoFlush, allows better use of a small cache when sessions are enabled. When sessions are enabled, deferred blocks that are written to the cache are locked and cannot be purged until an end_session call is made by the application. Once the cache has filled with deferred blocks no other blocks can be added. In order to keep the cache at peak performance the cache manager will now automatically issue a end_session call, this will write all deferred blocks to the disk, followed by a start_session call. By writing all the purged blocks to disk the cache manager can once again use it's LRU method to optimize performance.

(2) The Cache Manager now places itself in the "OutOfMemory" queue. If the memory manager cannot allocate the requested memory the Cache Manager will purge the GS/OS cache.

Device Driver Delta ERS

System Disk 5.0

Revision History

Date	Version	Who	Description
3/22/89	1.00	R. Montagne	Initial release.
	1.00+		Added warning about GS/OS Direct page.

Preface

This ERS outlines changes to the architecture of device drivers and supervisory drivers associated with performance enhancements provided by system disk 5.0 when switching between GS/OS applications and ProDOS 8 applications. These changes are optional. Existing drivers written for GS/OS under system disk 4.0 will remain compatible with system disk 5.0 without any changes although incorporating the changes described in this document will significantly improve system performance when switching between GS/OS applications and ProDOS 8 applications.

A new system call, D_RENAME, has been added to allow the renaming of devices. This call actually modifies the contents of the device driver's device information block in memory. A driver can be written to inhibit or enable the renaming function with respect to it's own device information block.

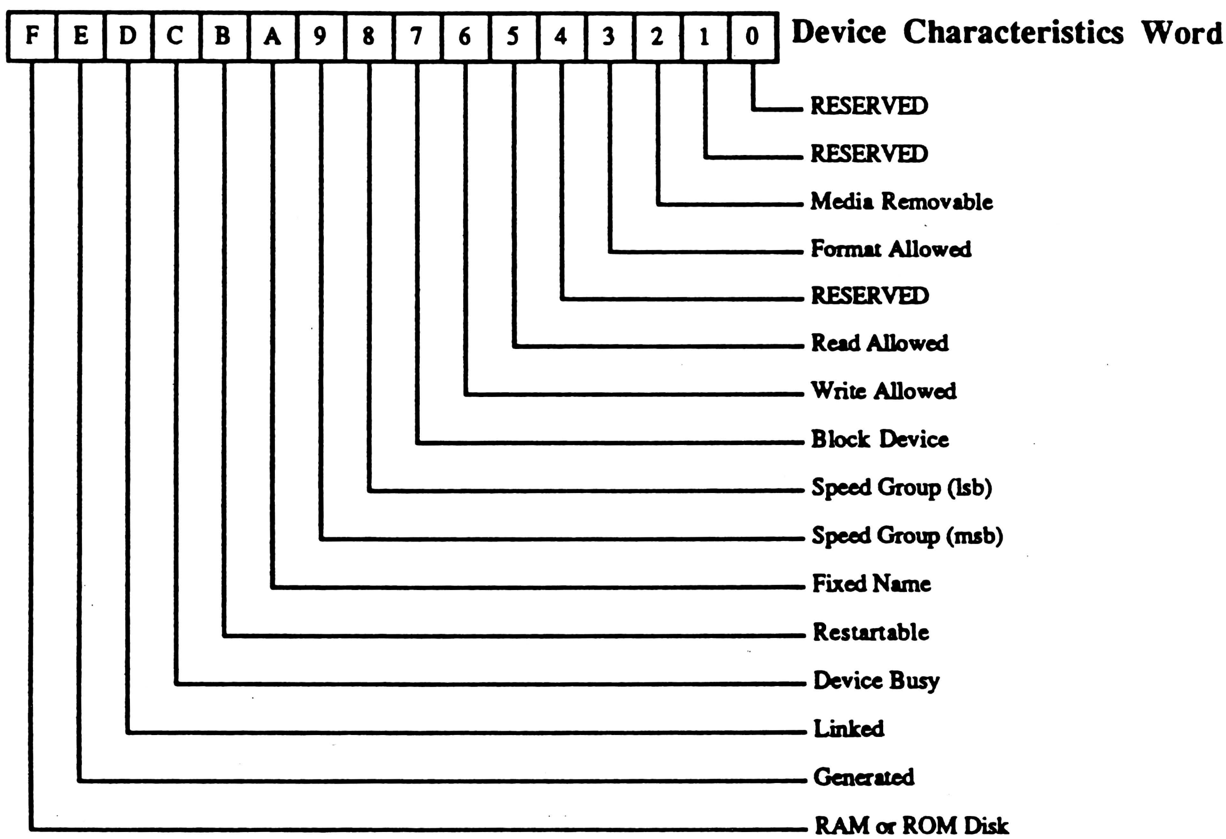
Background

Under system disk 4.0, GS/OS supported switching from GS/OS applications to ProDOS 8 applications and back. When switching back from ProDOS 8 applications to GS/OS applications a significant amount of time delay occurred because the GS/OS was reloaded from disk during this switch.

Under system disk 5.0 this time delay has been reduced significantly by keeping restartable components of the operating system in memory while running ProDOS 8 applications. When quitting from ProDOS 8 applications back to GS/OS the restartable components of the operating system are loaded from memory while the non restartable components are reloaded from disk. In an effort to maximize performance during the switch to GS/OS from a ProDOS 8 applications it is desirable to have as many components of the operating system revised to become restartable components as possible. In order to accomplish this goal the operating system supports restartable device drivers and supervisory drivers. All existing device drivers and supervisory drivers are considered non restartable.

Device Drivers

Two bits previously reserved in the device characteristics word contained in a device driver's device information block have now been defined to identify a driver as being renamable and/or restartable. If the 'Fixed Name' bit is set to a 1 then the D_RENAME call will not allow modification of the driver's name contained in the memory resident device information block. If the 'Restartable' bit is set to a 1 then the driver is considered to be restartable and will not be purged when quitting to a ProDOS 8 application from a GS/OS application. Restartable drivers are reloaded from memory when quitting from a ProDOS 8 application to a GS/OS application. The figure below depicts the new device characteristics word definition:



Note that all reserved fields in the supervisor information block and supervisor characteristics word must be cleared to 0.

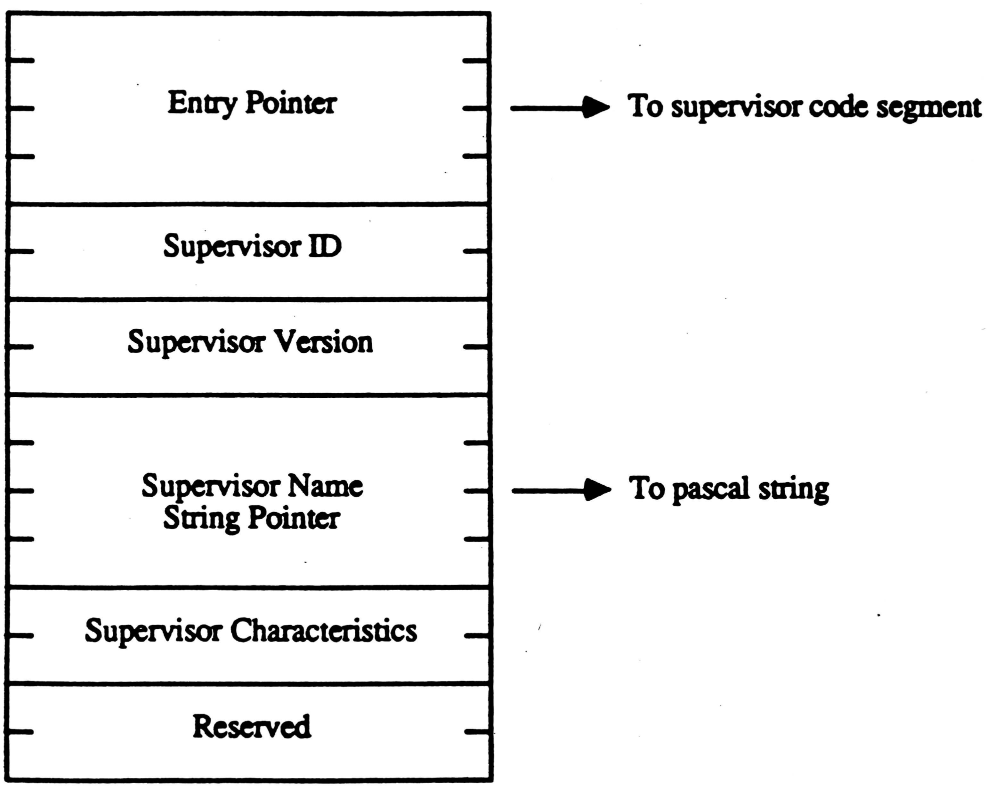
Making A Device Driver Restartable

Existing device drivers are always loaded from disk and thus may contain preinitialized data. This data may be modified during the normal execution of the device driver. In order to make these device drivers restartable, the device driver shutdown call must be modified to reinitialize the variables that have been modified during device driver execution so that a subsequent startup call to the

device driver will operate properly. This is an additional task for the device driver shutdown call and does not in any way diminish previous requirements on the device driver shutdown call.

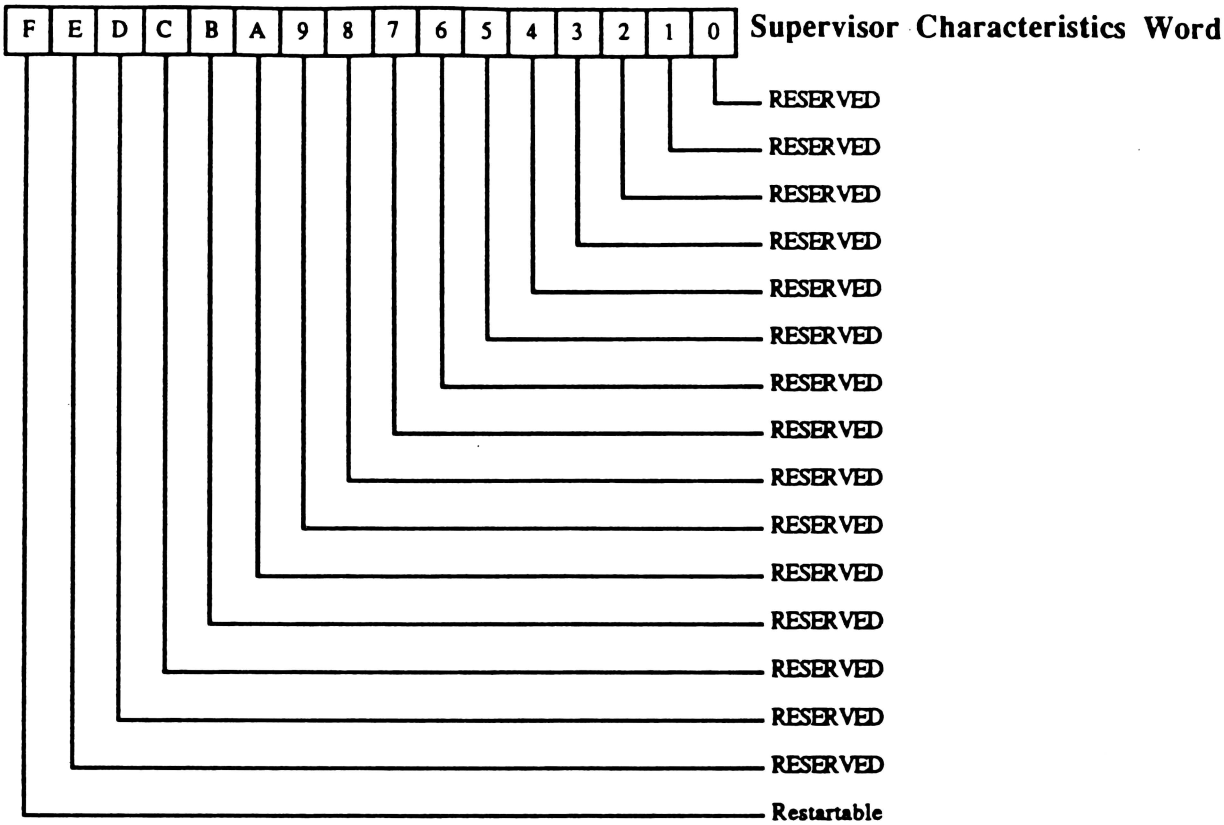
Supervisor Drivers

Several changes have been made to the supervisor driver information block. The contents of the supervisor information block on system disk 4.0 required a longword entry pointer, a supervisor ID word and a supervisor version word. This has now been expanded to include a longword pointer to a Pascal string containing the name of the supervisor driver and a supervisor characteristics word as shown below:



The supervisor name is optional. Supervisor drivers which do not support the supervisor name string must have a NIL pointer in the supervisor information block. Note that the supervisor name must follow the same syntax guidelines as device driver names (Upper case, no spaces, begin with an alpha, etc.).

The supervisor characteristics word describes whether a supervisor driver is restartable and is depicted in the figure below:



Note that all reserved fields in the supervisor information block and supervisor characteristics word must be cleared to 0.

Making A Supervisor Driver Restartable

Existing supervisor drivers are always loaded from disk and thus may contain preinitialized data. This data may be modified during the normal execution of the supervisor driver. In order to make these supervisor drivers restartable, the supervisor driver shutdown call must be modified to reinitialize the variables that have been modified during driver execution so that a subsequent startup call to the supervisor driver will operate properly. This is an additional task for the supervisor driver shutdown call and does not in any way diminish previous requirements on the supervisor driver shutdown call.

Warning

Contrary to previous documentation, no work space is provided on GS/OS direct page for either a device driver or supervisor driver. Do not use GS/OS direct page as a work space area under any circumstances. Damaged media could result from using GS/OS direct page as a work space. This warning holds true for system disk 4.0 and was previously distributed by developer technical services as soon as we discovered the anomaly. If you did not receive this vital piece of information previously, please heed the advice you are receiving now.

GS/OS Device Manager External ERS

Ver. 0.09a03

PRELIMINARY

Written by : Ray Montagne

Revision History

Date	Version	Description of Revision
03-30-1987	0.01	Initial release.
01-18-1988	0.02	D_INFO: Device link definition changed in DIB. D_STATUS: Device status now returns block count. D_STATUS: Device status word has additional bit definition for block devices.
02-07-88	0.01a01	Alpha release.
02-07-88	0.08a01	Update status call bit definition.
06-09-1988	0.09a01	New release. No technical changes.
21-04-1989	0.09a02	Updated information regarding D_Status & D_Control
05/11/1989	0.09a03	Added Forward to SCSI ERS for Partitions.

About the Device Manager

The Device Manager is similar to a File System Translator but is limited in it's support of GS/OS system calls. The Device Manager only supports GS/OS system calls that provide an application with direct access to a peripheral device or device driver that is not available through a File System Translator. The file system translator provides an interface between the application and the device dispatcher while maintaining compatibility with other components of GS/OS such as the Cache Manager and File System Translators.

Device Manager Calls

The device manager allows several calls to access the devices directly. Device manager calls include:

D_INFO
D_STATUS
D_CONTROL
D_READ
D_WRITE

Each of these calls will be described individually.

D_INFO: $2C

This call supports both class 0 (ProDOS16 Emulation Mode) and class 1 GS/OS system calls.

Class 0 Parameters:	Word	Input	Device Number
	Longword	Input	Pointer to output string

The class 0 D_INFO call returns the name of a device specified by device number in the buffer specified by the pointer to the output string.

Class 1 Parameters:	Word	Input	Parameter Count
	Word	Input	Device Number
	Longword	Input	Pointer to output string
	Word	Output	Device Characteristics
	Longword	Output	Total Blocks
	Word	Output	Slot Number
	Word	Output	Unit Number
	Word	Output	Version Number
	Word	Output	Device ID Number
	Word	Output	Head Device Link
	Word	Output	Forward Device Link

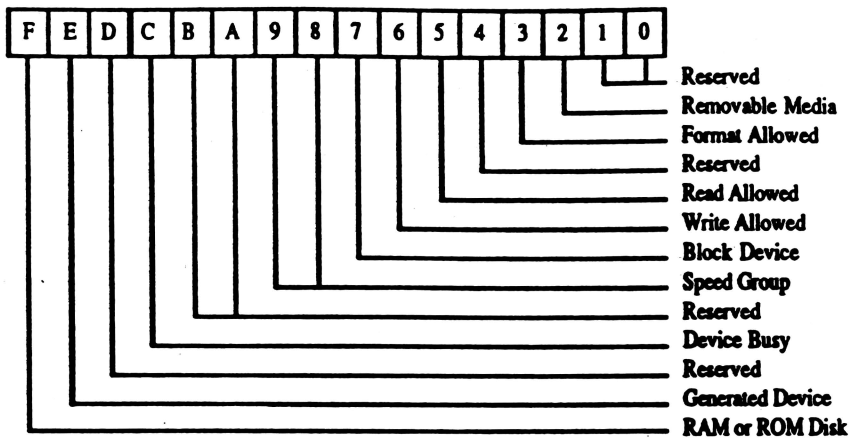
Parameter Count: This word parameter specifies the number of input and output parameters. The minimum value for this call is $0002.

Device Number: This word parameter specifies which device the call is to be issued to.

Device Name Pointer: This longword parameter points to the buffer that the device name will be returned in. For class 0 calls, the first byte of the buffer indicates the length of the device name followed by up to 31 bytes of ascii characters. For class 1 calls, the first word indicates the total size of the buffer while the second word indicates the length of the ascii string that follows.

Device Characteristics: This is a word parameter that describes features that may be implemented by the device. The definition of this word is shown in the figure below:

Device Characteristics



Total Blocks:	This is a longword parameter indicating the total number of blocks contained on a block device. This parameter has no application with character device drivers. Character device drivers will return with a value of zero.
Slot Number:	This word parameter specifies which slot the hardware or slot resident firmware associated with the driver resides in.
Unit Number:	This word parameter specifies which unit within a slot that the driver will access. This parameter has no correlation with device number.
Version Number:	This is a word parameter indicating the version number of the driver specified by the device number.
Device ID Number:	This word parameter is used to further identify a device. This parameter may be useful for 'FINDER' type applications when determining what type of ICON may be displayed for a particular device. Current definition of Device ID numbers include:

$0000 Disk][
$0001 Profile 5 Meg
$0002 Profile 10 Meg
$0003 Disk 3.5
$0004 SCSI (generic)
$0005 SCSI Hard Disk
$0006 SCSI Tape Drive
$0007 SCSI CD Rom
$0008 SCSI Printer
$0009 Serial Modem
$000A Console Driver
$000B Serial Printer
$000C Serial Laser Writer
$000D AppleTalk LaserWriter
$000E Ram Disk
$000F Rom Disk

$0010 File Server
$0011 IBX
$0012 Apple Desktop Bus
$0013 Hard Disk (generic)
$0014 Floppy Disk (generic)
$0015 Tape Drive (generic)
$0016 Character Device Driver (generic)
$0017 MFM Floppy Disk
$0018 Network (generic - AppleTalk?)
$0019 SCSI Sequential Access Device
$001A SCSI Scanner

None of the definitions shown in the table above indicate that any driver
or product related to a driver will be provided by Apple Computer, Inc.
The list has been provided in an attempt to consider what type of device
drivers may require OS support in the future. Apple Computer, Inc. is
responsible for assigning GS/OS device ID numbers. If you come up
with a device not listed Apple Computer, Inc. will assign a device ID
number for you.

Head Device Link: This word parameter describes the device number of the first device in a chain of linked devices. A value of NIL indicates that no link exists.

Forward Device Link: This word parameter describes the device number of the next device in a chain of linked devices. A value of NIL indicates that no link exists.

This call is used to obtain information contained in the Device Information Block (DIB) for a particular device. The device manager will make a call to the device dispatcher to obtain the pointer to the DIB. Then the device manager will return the requested parameters from the DIB. No actual dispatch to the device will occur.

D_STATUS: $2D

Class 0 parameters: There is no class 0 equivalent for this call.

Class 1 parameters:

Word	Input	Parameter Count
Word	Input	Device Number
Word	Input	Status Code
Longword	Input	Status List Pointer
Longword	Input	Request count
Longword	Output	Transfer Count

Parameter Count: Specifies the number of input and output parameters. The minimum value for this call is $0005.

Device Number: This word parameter specifies which device the call is to be issued to.

Status Code: This word parameter indicates which status call is to be made to the device specified by device number. Status codes of $0000 through $7FFF are standard status calls that must be supported by the device driver. Device specific status calls may be supported by a particular device. Device specific status calls use status codes $8000 through $FFFF. A list of standard status calls is shown below:

$0000	Device Status
$0001	Return Configuration Parameters
$0002	Wait / No Wait Status
$0003	Get Format Options
$0004	Get Partition Map (See SCSI ERS)
$0005 - $7FFF	Reserved by Apple Computer, Inc.
$8000 - $FFFF	Device Specific

Status List Pointer: This longword parameter points to a buffer that the status information is to be returned into. The application must allocate a buffer of adequate size for the status call being made.

Request Count: This longword indicates the number of bytes to be returned in the status list.

Transfer Count: This longword indicates the number of bytes that were returned in the status list.

The device manager does not need to set up any additional parameters to the device dispatcher for this call. This call is used to obtain specific status about a particular device. Each of the status calls is described individually.

Device Status:

The device status call returns a general device status word followed by a longword describing the total number of blocks on the device. This call requires a request count of $00000006. The bit definition within the general status word for character devices is as follows:

Bit 15	Reserved (currently read as zero)
Bit 14	1 = Linked Device
Bit 13	1 = Device Background Busy (executing a background task)
Bit 6-12	Reserved (currently read as zero)
Bit 5	Buffer not empty
Bit 4	1 = Online, 0 = Offline
Bit 2-3	Reserved (currently read as zero)
Bit 1	1 = Device currently interrupting
Bit 0	1 = Device currently open

The block count for character devices will be returned as zero.The bit definition within the general status word for block devices is as follows:

Bit 15	1 = Block count is uncertain for current block size
Bit 14	1 = Linked Device
Bit 13	1 = Device Background Busy (executing a background task)
Bit 5-12	Reserved (currently read as zero)
Bit 4	1 = Disk in drive, 0 = Disk not in drive
Bit 3	Reserved (currently read as zero)
Bit 2	1 = Write protected, 0 = Write enabled
Bit 1	1 = Device currently interrupting
Bit 0	1 = Disk has been switched

If either bit 0 or bit 4 has been set then the driver will call the SET_DISKSW via the system service call table.

Return Configuration Parameters:

This call returns a byte count as the first word in the status list which indicates the length of the Configuration parameter list in bytes. The Configuration parameters will be placed into the status list contiguous to the byte count. The structure of the Configuration parameter list is device dependent. The request count has a minimum of $00000002 to a maximum of $0000FFFF.

Wait / No Wait Status:

This call is used to determine if a device is in wait mode. A word value will be returned in the status list. If the word is set to a value of $0000, the device is operating in Wait mode. If the word is set to a value of $8000, the device is operating in No Wait mode. If in wait mode, the device will wait for the number of characters specified in the request count of a read call before returning from the read. An exception would be an early termination of the read call if a Newline character is encountered during the read. If in No Wait mode, a read call will return immediately with a transfer count indicating the number of characters returned. If a character was available, the transfer count will be returned from the read call with a non zero value. If a character was not available, the transfer count will be returned from the read call with a value of zero. Block device always

operate in Wait mode and will return a word with a value of $0000 in the status list. The request count for this call must be $00000002.

Get Format Options

This call returns a list of formatting options that may be selected using a Set_Format_Options call prior to issuing a format call to a block device. These parameters may include such variables as format environment, number of blocks, block size, and interleave. Devices that do not support media variables will return with a transfer count of zero and no error. The format of the status list on return from this call when a device does support media variables is as follows:

Returned List:	Word	Number of entries in list
	Word	Number of displayed entries
	Word	Recommended Default Option
	Word	Option that current online media is formatted with

Then each entry in the list consists of 16 bytes containing the following 5 fields:

Word	Media variables reference number
Word	Reference number of linked entry
Word	Flags
Long	Number of blocks supported by device
Word	Block Size
Word	Interleave Factor
Word	Format size (indicates block count * block size)

Bit definition within the flags word is as follows:

Bits 0 - 1	Format Type
Bits 2 - 3	Size multiplier
Bits 4 - 15	Reserved (must be zero)

Format type definitions are:

00	Universal Format
01	Apple Format
10	NonApple Format
11	Not valid

Size multipliers are:

00	Size in bytes
01	Size in K bytes
10	Size in M bytes
11	Size in G bytes

A typical list returned from this call for a device supporting two possible interleaves intended to support Apple's file systems (ProDOS, GS/OS, MFS or HFS) might be as follows:

Transfer count = $00000038 (56 bytes returned in list)

Returned List:	$0003	Three format options available
	$0002	Only two display entries
	$0001	Recommended default is option #1
	$0003	Current media is formatted as specified by option #3
	$0001	Refnum = Option #1
	$0002	LinkRef = entry #2
	$0005	Apple Format / size in kilobytes
	$00000640	Block count = 1600
	$0200	Block size = 512 bytes
	$0002	Interleave factor = 2:1
	$0320	Media size = 800 kilobytes
	$0002	Refnum = Option #2
	$0000	LinkRef = none
	$0005	Apple Format / size in kilobytes
	$00000640	Block count = 1600
	$0200	Block size = 512 bytes
	$0004	Interleave factor = 4:1
	$0320	Media size = 800 kilobytes
	$0003	Refnum = Option #3
	$0000	LinkRef = none
	$0005	Apple Format / size in kilobytes
	$00000320	Block count = 800
	$0200	Block size = 512 bytes
	$0004	Interleave factor = 4:1
	$0190	Media size = 400 kilobytes

Character devices should return no error.

D_CONTROL: $2E

Class 0 parameters: There is no class 0 equivalent for this call.

Class 1 parameters:

Word	Input	Parameter Count
Word	Input	Device Number
Word	Input	Control Code
Longword	Input	Control List Pointer
Longword	Input	Request Count
Longword	Input	Transfer Count

Parameter Count: This word parameter specifies the number of input and output parameters. The minimum value for this call is $0003.

Device Number: This word parameter specifies which device the call is to be issued to.

Control Code: This word parameter indicates which control call is to be made to the device specified by device number. Control codes of $0000 through $7FFF are standard control calls that must be supported by the device driver. Device specific control calls may be supported by a particular device. Device specific control calls use status codes $8000 through $FFFF. A list of standard control calls is shown below:

$0000	Reset Device
$0001	Format Device
$0002	Eject
$0003	Set Configuration Parameters
$0004	Set Wait / No Wait Mode
$0005	Set Format Options
$0006	Assign Partition Owner
$0007	Arm Signal
$0008	Disarm Signal
$0009	Set Partition Map (See SCSI ERS)
$000A - $7FFF	Reserved - these codes to be assigned by Apple Computer, Inc.
$8000 - $FFFF	Device Specific

Control List Pointer: This longword parameter points to a buffer that the control information is to be sent to a device or device driver.

Request Count: This longword indicates the number of bytes to be returned in the status list.

Transfer Count: This longword indicates the number of bytes that were returned in the status list.

The device manager does not need to set up any additional parameters to the device dispatcher for this call. This call is used to send control information to a particular device or device driver. Each of the control calls is described individually.

Reset Device:

This control call is used to set a device to it's default parameters. Reset requires a request count of $00000002.

Format Device:

This control call is used to format the media used by a block device. This call is not linked to any particular file system. It simply prepares all blocks on the media for reading and writing. Character devices do not support this function and return with no error. Format requires a request count of $00000000.

Eject:

This control call is used to physically eject the media from a block device. Character devices will implement this call by sending a form feed to the character device. Eject requires a request count of $00000000.

Set Configuration Parameters:

This control call is used to send device specific configuration parameters to a device. The first word in the control list indicates the length of the configuration parameter list in bytes. The configuration parameters should be placed into the control list contiguous to the byte count. The structure of the configuration parameter list is device dependent. *See the device driver ERS for more information*. The request count for this call depends on the existing configuration list for the device being accessed. The count must be equal to the existing count. A status call should be made to determine the current list size prior to issuing this call.

Wait / No Wait Mode:

This call is used to set a character device to Wait or No Wait mode. If in wait mode, the device will wait for the number of characters specified in the request count of a read call before returning from the read. An exception would be an early termination of input if a Newline character is encountered during the read. If in No Wait mode, a read call will return immediately with a transfer count indicating the number of characters returned. If a character was available the transfer count will return from a read with a value of $0001. If a character was not available the transfer count will be returned with a value of $0000. When making this call the control list should contain a word with a value of $0000 to set Wait Mode or a value of $8000 to set No Wait Mode. This control call has no application with block devices and should return with no error. Block devices only operate in Wait Mode. This call requires a request count of $00000002.

Set Format Options

This call is supported only by block devices and is used to set media specific parameters prior to executing a format call. This call does not imply a format. The control list consists of a word specifying a Format_RefNum and a word specifying the Interleave_Factor. The format reference number specifies a group of variables to be used during a subsequent format call which include Format Environment, Block Count, Block Size and Interleave Factor. If the Interleave_Factor is set to NIL then the default interleave specified in the format variables list will be used. In order to obtain a list of Format_RefNum values and their corresponding variables, a Get_Format_Options status call should be issued to the device. After the appropriate format variables have been selected, a Set_Format_Options control call should be issued followed by a format control call. This call is not supported by character devices and should return a 'BAD_COMMAND' error.

Control List: Word Format_RefNum

Word	Interleave_Factor

Assign Partition Owner

This call is supported by block devices and is intended to support partitioned media. This call is executed by an FST as a result of the system call 'ERASE_DISK'. The control list consists of a class 1 string indicating the partition owner. Partition names can be up to 32 bytes in length. Upper and lower case characters are considered equivalent. The driver will then reassign the current partition to the new owner. Block devices utilizing non-partitioned media and character devices should return with no error.

Control List:	String	Class 1 string specifying partition owner

Arm Signal

This call provides a means for a device driver to install an interrupt handler into the GS/OS signal manager's handler list. Signal code is an arbitrary value assigned by the driver to identify the signals that it generates. Each signal generated by the driver should have a unique signal code. Priority is the signal priority, with $0000 being the lowest priority and $FFFF being the highest priority. Handler address is the address where the signal handler resides.

Control List:	Word	Signal Code
	Word	Priority
	Longword	Signal Handler Address

Disarm Signal

This call provides a means for a device driver to remove it's interrupt handler from the GS/OS signal manager's handler list. Signal code is an arbitrary value assigned by the driver when the signal was armed.

Control List:	Word	Signal Code

D_READ: $2F

Class 0 parameters: There is no class 0 equivalent for this call.

Class 1 parameters:

Word	Input	Parameter Count
Word	Input	Device Number
Longword	Input	Buffer Pointer
Longword	Input	Request Count
Longword	Input	Starting Block
Word	Input	Block Size
Longword	Output	Transfer Count

Parameter Count: This word parameter specifies the number of input and output parameters. The minimum value for this call is $0006.

Device Number: This word parameter specifies which device the call is to be issued to.

Buffer Pointer: This longword parameter specifies a buffer in memory that the data read from the device is to be written into. The application must allocate a buffer of adequate size to accommodate the data.

Request Count: This longword parameter specifies the number of bytes to be read from the device and placed into the buffer specified by buffer pointer.

Starting Block: This longword parameter specifies the logical block address on a block device that data is to be read from. In a multiple block read, this parameter specifies the first logical block to be read. This parameter has no application with character device drivers.

Block Size: This word parameter specifies the size of the block addressed by the block number. This parameter must be a non-zero value for block devices. Block devices interpret a value of zero literally. This parameter must be set to zero for character devices.

Transfer Count: This is a longword parameter returned by the call that indicates the number of bytes actually transferred.

The device manager must determine if this call is being made to a block or character device. The device manager will make a call to the device dispatcher to obtain a pointer to the DIB for this purpose. The device characteristics word in the DIB will be examined to determine if the device is a character device or a block device. If the call is directed to a character device, the device manager need only set the block size parameter to a value of $0000 prior to dispatching to the device via the device dispatcher. If the call is directed to a block device, the device manager must set the FST number, volume ID and cache priority before dispatching to the device via the device dispatcher. The FST number, volume ID and cache priority will always be set to a value of $0000. This forces access to the media while inhibiting the caching function.

This call is used to transfer data from the device to the buffer specified in the parameter block on direct page. A character device must be opened prior to conducting I/O transactions with that device. A standard file open call must be issued via the character FST to the device to meet this requirement. If an I/O transaction is attempted with a device that has not been opened, the driver should return a 'NOT_OPEN' error. The driver will set the transfer count to zero prior to performing an I/O transaction with the device. The transfer count will then be incremented by the driver as each byte is received from the device. Character devices must support the Newline function. As each character received from the device is placed into the buffer, the character should be ANDed with the Newline character mask and compared to the Newline character list. If a compare results, the I/O transaction should be terminated with no error and the transfer count set to the number of bytes written to the buffer. If a Newline character is not encountered, the I/O transaction should be terminated when the transfer count equals the request count.

Block devices do not have the requirement of being 'opened' prior to conducting the I/O transaction. The block of data specified by block number will be returned into the buffer specified by buffer pointer. The block will always be read from the device, not the cache. In addition, no caching operation will take place on the block being read.

D_WRITE: $30

Class 0 parameters:	There is no class 0 equivalent for this call.

Class 1 parameters:

Word	Input	Parameter Count
Word	Input	Device Number
Longword	Input	Buffer Pointer
Longword	Input	Request Count
Longword	Input	Starting Block
Longword	Input	Block Size
Longword	Output	Transfer Count

Parameter Count: This word parameter specifies the number of input and output parameters.. The minimum value for this call is $0006.

Device Number: This word parameter specifies which device the call is to be issued to.

Buffer Pointer: This longword parameter specifies a buffer in memory that the data read from the device is to be written into. The application must allocate a buffer of adequate size to accommodate the data.

Request Count: This longword parameter specifies the number of bytes to be read from the device and placed into the buffer specified by buffer pointer.

Starting Block: This longword parameter specifies the logical block address on a block device that data is to be read from. In a multiple block read, this parameter specifies the first logical block to be read. This parameter has no application with character device drivers.

Block Size: This word parameter specifies the size of the block addressed by the block number. This parameter must be a non-zero value for block devices. Block devices interpret a value of zero literally. This parameter must be set to zero for character devices.

Transfer Count: This is a longword parameter returned by the call that indicates the number of bytes actually transferred.

The device manager must determine if this call is being made to a block or character device. The device manager will make a call to the device dispatcher to obtain a pointer to the DIB for this purpose. The device characteristics word in the DIB will be examined to determine if the device is a character device or a block device. If the call is directed to a character device, the device manager need only set the block size parameter to a value of $0000 prior to dispatching to the device via the device dispatcher. If the call is directed to a block device, the device manager must set the FST number, volume ID and cache priority before dispatching to the device via the device dispatcher. The FST number, volume ID and cache priority will always be set to a value of $0000. This inhibits the caching function.

This call is used to transfer data to the device from the buffer specified in the parameter block . A character device must be opened prior to conducting I/O transactions with that device. A standard file open call must be issued via the character FST to the device to meet this requirement. If an I/O transaction is attempted with a device that has not been opened, the driver should return a 'NOT-OPEN' error. The driver will set the transfer count to zero prior to performing an I/O transaction with the device. The transfer count will then be incremented by the driver as each byte is written to the device.

Block devices do not have the requirement of being 'opened' prior to conducting the I/O transaction. The block of data specified by block number will be written from the buffer specified by buffer pointer. In addition, no caching operation will take place on the block being written.

GS/OS Disk][Driver ERS

Ver. 2.01d12

Alpha Series Release

Written by : Ray Montagne

Revision History

Date	Version	Description of Revision
28 Aug 87	0.01e01	Initial release.
09 Feb 88	0.01a01	First alpha release.
19 May 88	0.08a01	Page three now indicates support for fourteen Disk][devices.
06-09-1988	0.09a01	New release. No technical changes.
01 Jan 1989	4.10a01	Device status update. Format options was incorrect.
22 Feb 1989	2.01d12	Changed ers version to match disk. References to DEVICE MANAGER executing retries on disk switched errors have been removed.

About This Document

This document is intended to describe the GS/OS Disk][driver calls. Included is a brief description of how each call is implemented with respect to the Disk][. Also included is a description of the physical and logical formats used with the Disk][media.

General Information

The Disk][driver is a loaded driver which does not require a supervisory driver to conduct I/O transactions with the Disk][device. It supports up to fourteen Disk][devices and should operate equally well with a Disk][Interface Card or IWM interface. The Disk][driver operates independent of the system speed and does not have the resident slot limitation inherent in the Apple//GS. (The Apple//GS normally only allows Disk][devices in slots 4 through 7 in fast mode. This driver operates with Disk][devices in slots 1 through 7 in fast mode with either one or two Disk][devices per slot.)

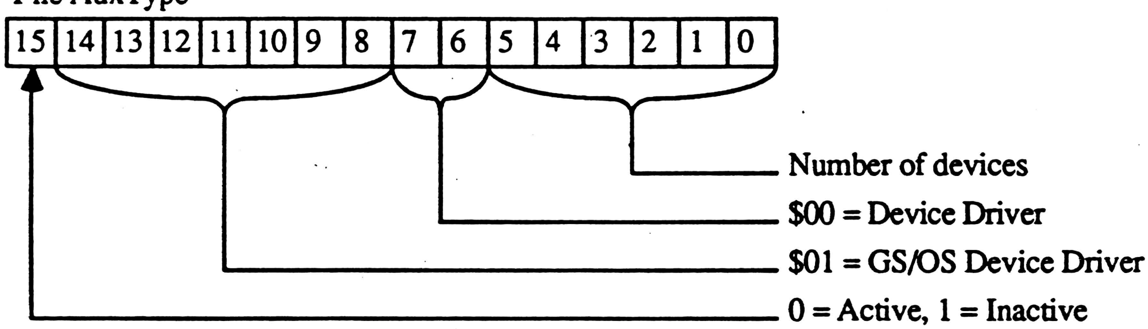
Disk][Limitations

The Disk][device provides no means for detection of disk switched errors. A simulation of disk switched is provided that will force any file system translator interfacing to the Disk][to identify the volume currently online. Simulation of disk switched errors is adequate to force volume identification but is not adequate to validate the integrity of the cache. For this reason, the Disk][driver does not implement caching. Additionally, a status call will never return a disk switched status.

Disk][Driver FileType and AuxType

The Disk][driver is compacted and has a filetype of $BB as do all GS drivers. The AuxType for the Disk][driver has been set to $010E indicating that the driver is a GS/OS driver supporting a maximum of 14 ($0E) devices.

File AuxType



Device Driver Structure

The Disk][driver consists of a driver header, configuration parameter list, device information block and the driver code segment. No scripts exist for the Disk][driver, therefore, no script segment has been provided. No configuration parameters exist for the Disk][driver. The configuration parameter list for each Disk][device has a length word of NIL.

About the Driver Header

The header is used when loading the driver. It indicates where the configuration parameter lists and DIBs are located. The device dispatcher loads only the driver, DIBs and configuration parameter lists using an initial_load call to the system loader. The header contains the following information:

Word	Offset to 1st DIB
Word	Count of number of devices = 14
Word	Offset to 1st configuration parameter list for device #1
Word	Offset to 1st configuration parameter list for device #2
Word	Offset to 1st configuration parameter list for device #3
Word	Offset to 1st configuration parameter list for device #4
Word	Offset to 1st configuration parameter list for device #5
Word	Offset to 1st configuration parameter list for device #6
Word	Offset to 1st configuration parameter list for device #7
Word	Offset to 1st configuration parameter list for device #8
Word	Offset to 1st configuration parameter list for device #9
Word	Offset to 1st configuration parameter list for device #10
Word	Offset to 1st configuration parameter list for device #11
Word	Offset to 1st configuration parameter list for device #12
Word	Offset to 1st configuration parameter list for device #12
Word	Offset to 1st configuration parameter list for device #14

About the DIB

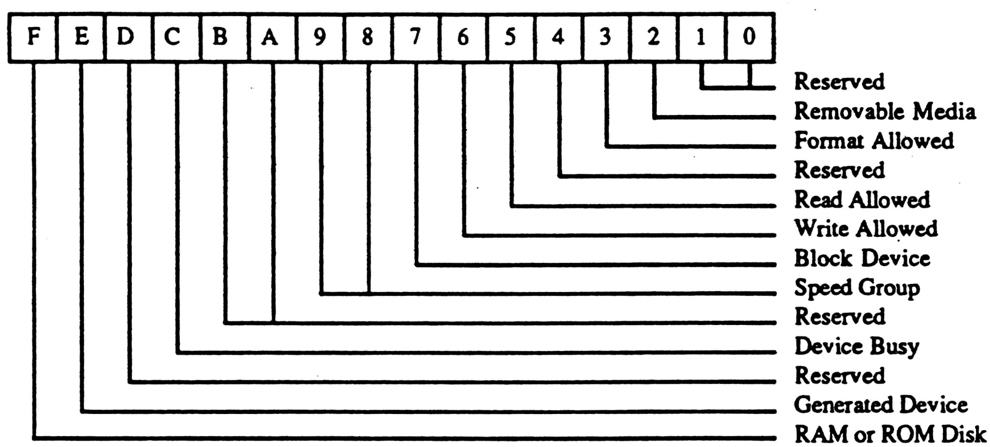
The DIB must contain the following information:

The link pointer is a longword pointing to the next DIB for device drivers supporting multiple DIBs. If the device driver supports only a single DIB then the link pointer should be set to NIL. This is used to install the device drivers into the device list. This pointer does not imply any links between devices.

The entry pointer is a longword pointer to the device driver's entry point. The Disk][driver has a common entry point referenced by the DIB for each Disk][device.

The device characteristics word parameter describes features that may or may not be supported by the device. The Disk][driver's device characteristics are set to $03EC indicating that the Disk][is not speed dependent, is a block device and supports removable media, formatting, read and write operations. A pictorial representation of the device characteristics word is shown below:

Device Characteristics



Definition of the Speed Group bits are as follows:

00	1 Mhz Device
01	2.6 Mhz Device
10	>2.6 Mhz Device
11	Device is not speed dependent

Speed independence is achieved through use of the system service call "set_speed" prior to executing time critical sections of the code.

The Block Count is a longword parameter which is only used with block devices. It indicates the total number of blocks accessible on the device. The Disk][driver indicates the number of 512 byte blocks in the DIB as 280 blocks.

Device Name is a 32 byte field which contains a count byte followed by a device name encoded in up to 31 bytes of ASCII. Note that the initial '.' is not included in the device name. The device name must be in upper case with the MSB off. The Disk][driver returns a seven character name "DISK][x" where "x" is a unique character for each Disk][device.

Slot Number is the number of the slot where the device hardware resides. Bits 0 through 2 indicate the slot while bit 3 indicates that the slot is internal or external.

Unit Number is the device number within the slot. This is not a global unit number relating to the device list.

Device Version Number is a word parameter which indicates the version number of either a loaded or generated driver. Generated drivers may use the version number obtained from the slot resident firmware interface. The most significant nibble of the version indicates the major release version while the next two most significant nibbles indicate the minor release version. The least significant nibble indicates: E=Experimental, A=Alpha, B=Beta, 0=Final. The first release of the Disk][driver has a version of $001A indicating a first alpha phase release.

Device ID Number is a word specifying the type of device. The Disk][driver has a device id of $0000.

The Disk][driver does not support linked devices. Both the Head and Forward links in the DIB are set to a NIL value.

Two additional words have been reserved in the DIB for future expansion. These words are set to NIL.

Disk][Media

The Disk][driver supports only 35 track 16 sector media. Media is formatted with a physical 1:1 interleave. Logical interleave is achieved by using one of two interleave translation tables. Dos operates on 256 byte sectors. ProDOS and Pascal operate on 512 byte blocks consisting of two contiguous logical sectors. Both ProDOS and Pascal use a common logical sector interleave of 2:1 while Dos uses a logical sector interleave of 14:1. Interleave translation table selection is based on the block size set as input to a media access call. Logical to physical sector translations are shown in the figures below:

Disk][Interleave as used by ProDOS:

LOGICAL SECTOR ADDRESS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
PHYSICAL SECTOR ADDRESS	0	2	4	6	8	A	C	E	1	3	5	7	9	B	D	F

Disk][Interleave as used by Pascal:

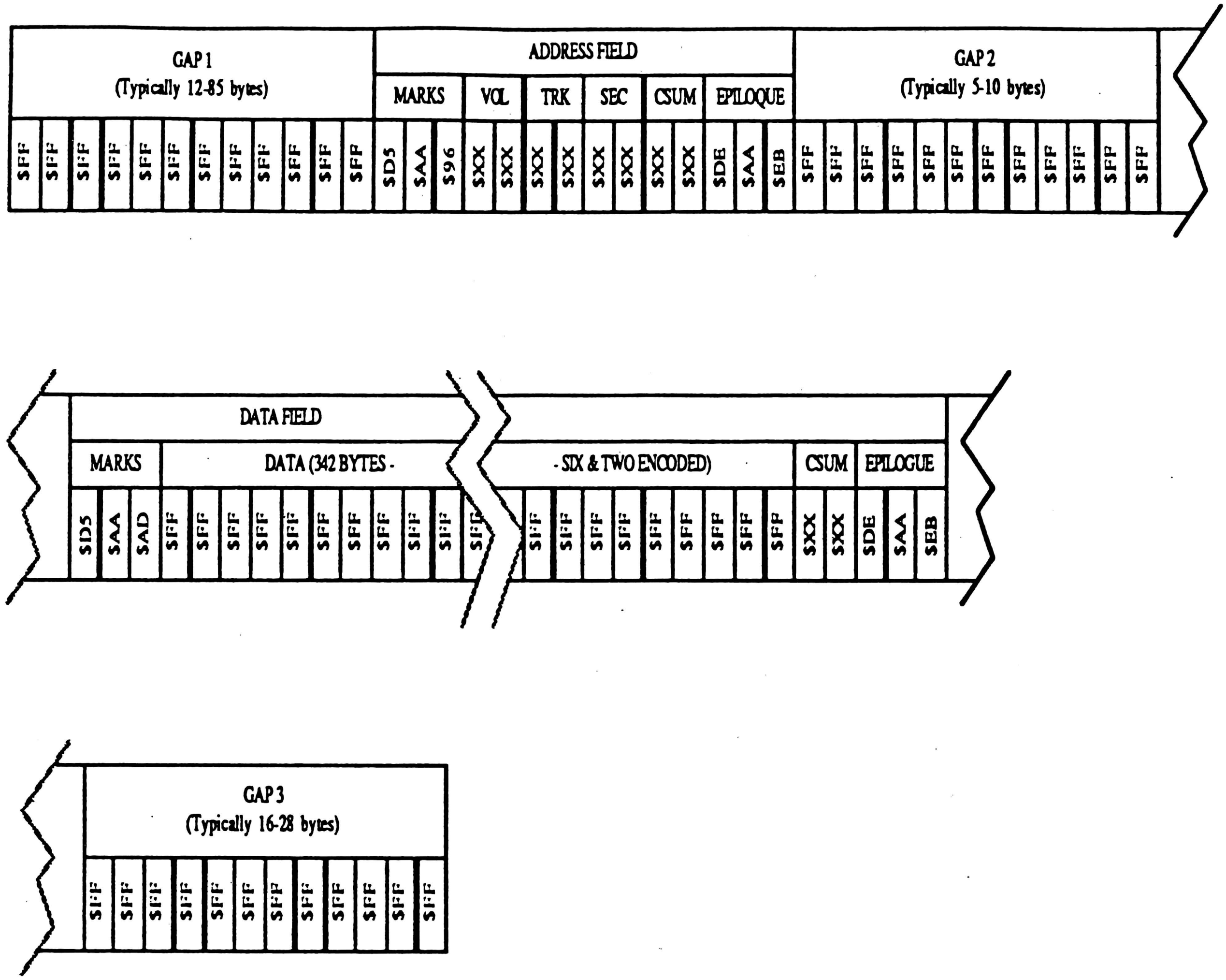
LOGICAL SECTOR ADDRESS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
PHYSICAL SECTOR ADDRESS	0	2	4	6	8	A	C	E	1	3	5	7	9	B	D	F

Disk][Interleave as used by DOS3.3:

LOGICAL SECTOR ADDRESS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
PHYSICAL SECTOR ADDRESS	0	D	B	9	7	5	3	1	E	C	A	B	6	4	2	F

Each sector consists of a self synchronization gap, address field, second self synchronization gap, data marks and data field as shown in the figure below.

DISK][SECTOR FORMAT



How the Disk][Driver is called

Applications may access the Disk][device either through a file system translator (such as ProDOS, Pascal or DOS3.3) or via the Device Manager calls (D_INFO, D_READ or D_WRITE).

Disk][Driver Calls

All drivers will accept a standard set of calls. Disk][driver calls accessible to an FST or the Device Manager will include :

- DRIVER_OPEN (FST only)
- DRIVER_READ
- DRIVER_WRITE
- DRIVER_CLOSE (FST only)
- DRIVER_STATUS
- DRIVER_CONTROL
- DRIVER_FLUSH (FST only)

The details of each driver call will be described individually. Each of these calls will be described in detail on the following pages.

Drvr_Open

Call Parameters :	Device Number	≠ $0000
	Call Number	= $0001
	DIB Pointer	

Device Number: This word parameter specifies which device is to be accessed by the call. This parameter must be a non-zero value.

Call Number: This word parameter specifies which type of call is to be issued to the device.

DIB Pointer: This longword points to the device information block for the device being accessed.

This call has no function with block devices. The Disk][driver will return with no error.

Drvr_Read

Call Parameters :	Device Number	≠ \$0000
	Call Number	= \$0002
	Buffer Pointer	
	Request Count	
	Transfer Count	
	Block Size	≠ \$0000
	FST Number	
	DIB Pointer	

Device Number: This word parameter specifies which device is to be accessed by the call. This parameter must be a non-zero value.

Call Number: This word parameter specifies which type of call is to be issued to the device.

Buffer Pointer: This is a longword pointer to memory where the data is to be written to after being read from the device.

Request Count: This is a longword specifying the number of bytes that the driver is being requested to transfer from the device to the buffer specified by buffer pointer.

Transfer Count: This is a longword returned by the call that indicates the number of bytes actually transferred.

Block Number: This longword parameter specifies the logical address within the block device from which data is to be transferred from. This parameter has no application in character device drivers.

Block Size: This word parameter specifies the size of the block addressed by the block number. This parameter must be a non-zero value for block devices. This parameter must be set to a value of zero for character devices.

FST Number: This word parameter specifies the File System Translator that owns the volume for which the block is being transferred.

DIB Pointer: This longword points to the device information block for the device being accessed.

This call returns the requested number of bytes from the disk starting at the block number specified. The request count must be an integral multiple of the block size. If during a multiple block transaction the block address exceeds the block address range for the Disk][then a bad block error will be returned with the transfer count indicating the number of bytes read successfully from the Disk][device. Note that the Disk][driver supports a block size of 256 bytes or 512 bytes and block counts of 560 and 280 blocks respectively. Logical interleaving on the disk varies with the block size. It should also be noted that if the Disk][has not had a media access call in the one second previous to issuing this call then a disk switched error will be returned except if the call is issued through the device manager.

Driver Write

Call Parameters :	Device Number	≠ \$0000
	Call Number	= \$0003
	Buffer Pointer	
	Request Count	
	Transfer Count	
	Block Size	≠ \$0000
	FST Number	
	DIB Pointer	

Device Number: This word parameter specifies which device is to be accessed by the call. This parameter must be a non-zero value.

Call Number: This word parameter specifies which type of call is to be issued to the device.

Buffer Pointer: This is a longword pointer to memory where the data is to be written to after being read from the device.

Request Count: This is a longword specifying the number of bytes that the driver is being requested to transfer from the device to the buffer specified by buffer pointer.

Transfer Count: This is a longword returned by the call that indicates the number of bytes actually transferred.

Block Number: This longword parameter specifies the logical address within the block device from which data is to be transferred from. This parameter has no application in character device drivers.

Block Size: This word parameter specifies the size of the block addressed by the block number. This parameter must be a non-zero value for block devices. This parameter must be set to a value of zero for character devices.

FST Number: This word parameter specifies the File System Translator that owns the volume for which the block is being transferred.

DIB Pointer: This longword points to the device information block for the device being accessed.

This call write the requested number of bytes to the disk starting at the block number specified. The request count must be an integral multiple of the block size. If during a multiple block transaction the block address exceeds the block address range for the Disk][then a bad block error will be returned with the transfer count indicating the number of bytes read successfully from the Disk][device. Note that the Disk][driver supports a block size of 256 bytes or 512 bytes and block counts of 560 and 280 blocks respectively. Logical interleaving on the disk varies with the block size. It should also be noted that if the Disk][has not had a media access call in the one second previous to issuing this call then a disk switched error will be returned except if the call is issued through the device manager.

Driver Close

Call Parameters :	Device Number	≠ $0000
	Call Number	= $0004
	DIB Pointer	

Device Number: This word parameter specifies which device is to be accessed by the call. This parameter must be a non-zero value.

Call Number: This word parameter specifies which type of call is to be issued to the device.

DIB Pointer: This longword points to the device information block for the device being accessed.

This call has no function with block devices. The Disk][driver will return with no error.

Driver Status

Call Parameters :	Device Number	≠ $0000
	Call Number	= $0005
	Status List Pointer	
	Request Count	
	Transfer Count	
	Status Code	
	DIB Pointer	

Device Number: This word parameter specifies which device is to be accessed by the call. This parameter must be a non-zero value.

Call Number: This word parameter specifies which type of call is to be issued to the device.

Status List Pointer: This is a longword pointer to memory where the status list is to be written into.

Request Count: This longword parameter passed to the call indicates the number of bytes to be transferred. If the request count is smaller than the minimum buffer size required by the call, an error will be returned.

Transfer Count: This is a longword returned by the call that indicates the number of bytes actually transferred.

Status Code: This is a word parameter specifying the type of status request. Status codes of $0000 through $7FFF are standard status calls that must be supported by device drivers. Devices supporting device specific status calls should use status codes in the range of $8000 through $FFFF. A list of standard status calls is shown below:

$0000	Device Status
$0001	Return Configuration Parameters
$0002	Return Wait / No Wait Status
$0003	Get Format Options
$0004 - $7FFF	Reserved - these status codes to be assigned by Apple Computer, Inc.
$8000 - $FFFF	Device Specific

DIB Pointer: This longword points to the device information block for the device being accessed.

This call is used to obtain current status information from the device or the driver. Extensions to the standard set of calls which transfer data or status information from the device or device driver are supported as a subset of this call. The device driver is responsible for validating the status code prior to executing the requested status call. If an invalid status code is passed to the driver, the driver should return a 'BAD CODE' error. The device dispatcher will set the transfer count to zero prior to calling the device driver. The device driver should set the transfer count to the number of bytes returned as a result of the status call.

Device Status

This call returns a general status followed by a longword specifying the number of blocks supported by the device. Write protect reflects the state of the write protect sense line on the previous media access.

Bit 15	0 = Block count certain, 1 = Block count uncertain for block size
Bit 14	0
Bit 5-13	Reserved (currently read as zero)
Bit 4	1 = Disk in drive, 0 = Disk not in drive
Bit 3	Reserved (currently read as zero)
Bit 2	1 = Write protected, 0 = Write enabled
Bit 1	1 = Device currently interrupting
Bit 0	1 = Disk has been switched

Note that there is no way to validate media insertion on a Disk][. Bit 4 of the device status word will always be set to a '1'.

Status List:	Word	General status word
	Long	Number of blocks supported by device

Return Configuration Parameters

This call returns a byte count as the first *word* in the status list which indicates the length of the configuration parameter list in bytes. The configuration parameters will be placed into the status list contiguous to the byte count. The Disk][has no parameters in it's configuration parameter list and will return with a length word of zero and transfer count of $00000002.

Status List:	Word	Length of configuration parameter list
	Data	Data returned from configuration parameter list

Wait / No Wait Status

Block devices only operate in WAIT mode. This call always returns a word of $0000 and a transfer count of $00000002 for Disk][devices.

Status List:	Word	Wait status

Get Format Options

This call returns a list of formatting options that may be selected using a Set_Format_Options call prior to issuing a format call to a block device. These parameters may include such variables as format environment, number of blocks, block size, and interleave. Devices that do not support media variables will return with a transfer count of zero and no error. The format of the status list on return from this call when a device does support media variables is as follows:

Returned List:	Word	Number of entries in list
	Word	Number of displayed entries in list
	Word	Recommended Default Option
	Word	Option that current online media is formatted with

Then each entry in the list consists of 16 bytes containing the following 5 fields:

Word	Media variables reference number
Word	Reference number of linked entry
Word	Flags
Long	Number of blocks supported by device
Word	Block Size
Word	Interleave Factor
Word	Media size (block count * block size)

Flags word definition is as follows:	Bits 0 - 1	Format Type
	Bits 2 - 3	Size Multiplier
	Bits 4 - 15	Reserved (must be zero)

Format Type definition is as follows:	00	Universal Format
	01	Apple Format
	10	NonApple Format
	11	Not valid

Size Multiplier definition is as follows:	00	Size in bytes
	01	Size in kilobytes
	10	Size in megabytes
	11	Size in gigabytes

The Disk][driver returns format options as follows:

Transfer count = $00000028 (40 bytes returned in list)

Returned List:	$0002	Two entries in list
	$0001	Only one display entries
	$0001	Recommended default is option #1
	$0000	Current media formatted is unknown
	$0001	Refnum = Option #1
	$0002	LinkRef = Option #2
	$0004	Universal format / size in kilobytes
	$00000118	Block count = 280
	$0200	Block size = 512 bytes
	$0000	Interleave factor = n/a (fixed physical interleave)
	$008C	Media size = 140 kilobytes
	$0002	Refnum = Option #2
	$0000	LinkRef = NIL
	$0004	Universal format / size in kilobytes
	$00000230	Block count = 560
	$0100	Block size = 256 bytes
	$0000	Interleave factor = n/a (fixed physical interleave)
	$008C	Media size = 140 kilobytes

Control Call

Call Parameters :	Device Number	≠ \$0000
	Call Number	= \$0006
	Control List Pointer	
	Request Count	
	Transfer Count	
	Control Code	
	DIB Pointer	

Device Number: This word parameter specifies which device is to be accessed by the call. This parameter must be a non-zero value.

Call Number: This word parameter specifies which type of call is to be issued to the device.

Control List Pointer: This is a longword pointer to memory where the control list is to be read from.

Request Count: This longword parameter passed to the call indicates the number of bytes to be transferred. If the request count is smaller than the minimum buffer size required by the call, an error will be returned.

Transfer Count: This is a longword returned by the call that indicates the number of bytes actually transferred.

Control Code: This is a word parameter specifying the type of control request. Status codes of \$0000 through \$7FFF are standard control calls that must be supported by device drivers. Devices supporting device specific control calls should use control codes in the range of \$8000 through \$FFFF. A list of standard control calls is shown below:

\$0000	Reset Device
\$0001	Format Device
\$0002	Eject
\$0003	Set Configuration Parameters
\$0004	Set Wait / No Wait Mode
\$0005	Set Format Options
\$0006	Assign Partition Owner
\$0007	Arm Event
\$0008	Disarm Event
\$0009 - \$7FFF	Reserved - these status codes to be assigned by Apple Computer, Inc.
\$8000 - \$FFFF	Device Specific

DIB Pointer: This longword points to the device information block for the device being accessed.

This call is used to send control information to the device or the device driver. Extensions to the standard set of calls which transfer data or control information to the device or device driver are supported through the use of device specific control codes.

The device driver is responsible for validating the control code and control list length prior to executing the requested control call. If an invalid control code is passed to the driver, the driver should return a 'BAD CODE' error. If an invalid control list length is passed to the driver, the

driver should return a 'BAD PARAMETER' error. The device driver should set the transfer count to the number of bytes processed as a result of a successful control call.

Reset Device

This control call is used to reset a particular device to it's default settings. This call has no function with the Disk][driver and returns with no error.

Control List:	Word	Length of control list ($0000)

Format Device

This control call is used to format the media used by a block device. This call is not linked to any particular file system. It simply prepares all blocks on the media for reading and writing.

Control List:	Word	Length of control list ($0000)

Eject

The Disk][device does not have any mechanism apart from the user's hand for ejecting media. This call has no function with the Disk][driver and returns with no error.

Control List:	Word	Length of control list ($0000)

Set Configuration Parameters

This call has no function with the Disk][driver and returns with no error.

Control List:	Word	Length of configuration parameter list ($0000)
	Data	Configuration Parameter List Data

Wait / No Wait Mode

All block devices including the Disk][operate in WAIT mode only. Setting the Disk][driver to wait mode results in no error. If a call is issued to set the Disk][driver to no wait mode then a bad parameter error will be returned.

Control List:	Word	Length of control list ($0002)
	Word	Wait / No Wait Mode

Set Format Options

This call has no function with the Disk][driver since only a single fixed physical interleave is supported. This call returns with no error.

Control List:	Word	Length of control list ($0004)
	Word	Format_RefNum
	Word	Interleave_Factor

Assign Partition Owner

This call has no function with the Disk][driver and returns with no error.

Control List:	String	Class 1 string specifying partition owner

Arm Signal

This call has no function with the Disk][driver and returns with no error.

Control List:	Word	Signal Code
	Word	Priority
	Longword	Signal Handler Address

Disarm Signal

This call has no function with the Disk][driver and returns with no error.

Control List:	Word	Signal Code

Driver Flush

Call Parameters :	Device Number	≠ $0000
	Call Number	= $0007
	DIB Pointer	

Device Number: This word parameter specifies which device is to be accessed by the call. This parameter must be a non-zero value.

Call Number: This word parameter specifies which type of call is to be issued to the device.

DIB Pointer: This longword points to the device information block for the device being accessed.

This call has no function with the Disk][driver and returns with no error.

Device Driver Error Codes

All error codes listed below must be supported by device drivers wherever applicable. All block device drivers must support disk switched errors without exception. Please take note that the error codes are returned from a device driver must have the high byte cleared. The device dispatcher maintains certain error codes under certain conditions. Device dispatcher error codes are passed in the upper byte of the accumulator.

Error Code	Description	Mnemonic
$0000	No error occurred	NO_ERROR
$0010	Device not found	DEV_NOT_FOUND
$0011	Invalid Device Number	INVALID_DEV_NUM
$0020	Invalid request	DRVR_BAD_REQ
$0021	Invalid control or status code	DRVR_ BAD_CODE
$0022	Invalid parameter	DRVR_ BAD_PARM
$0023	Device not open (character driver only)	DRVR_ NOT_OPEN
$0024	Device already open (character driver only)	DRVR_ PRIOR_OPEN
$0026	Resource not available	DRVR_ NO_RESRC
$0027	I/O error	DRVR_ IO_ERROR
$0028	Device not connected	DRVR_ NO_DEV
$0029	Device is busy	DRVR_BUSY
$002B	Write Protected (block driver only)	DRVR_ WR_PROT
$002C	Invalid Byte Count	DRVR_ BAD_COUNT
$002D	Invalid Block Number (block driver only)	DRVR_ BAD_BLOCK
$002E	Disk Switched (block driver only)	DRVR_ DISK_SW
$002F	Device Off Line or No Media Present	DRVR_ OFF_LINE
$004E	Invalid access or access not allowed	INVALID_ACCESS
$0058	Not a block device	NOT_BLOCK_DEV
$0060	Data is unavailable	DATA_UNAVAIL

ExpressLoad™

version 0.03

By
Rob Turner

Revision History:

version 0.01 11/30/88	first version.
version 0.02 02/22/89	added detailed example.
version 0.03 02/27/89	added corrections.

Introduction:

This document describes the new piece of system software called ExpressLoad™ and how it will affect the system and applications. ExpressLoad is a piece of system software that compliments the standard Apple IIGS system loader by allowing large applications to be loaded in a shorter period of time. To accomplish the faster loading, the application has to be stored on disk in ExpressLoad format (described later in this document).

Overview:

ExpressLoad works as a "front end" to the current system loader. Its use is invisible to applications that use the system loader. As a front end, ExpressLoad will automatically determine if the file being loaded is in ExpressLoad format. If the file is in ExpressLoad format then ExpressLoad will process the file without passing control to the system loader. If however, the file is not in ExpressLoad format, control will be passed onto the system loader, where the file will be processed as normal. Note: A file in ExpressLoad format can still be loaded by the system loader.[1]

ExpressLoad achieves better performance by applying the following techniques to the file and loading of the application:

- Moving all segment headers into the ExpressLoad segment data area.
- Keeping the ExpressLoad segment in memory.
- Knowing the position and size of each segment in the file, including the relocation dictionary.
- Converting each code segment to one LConst record. (eliminates double buffering and the need for a general work buffer)
- Making the "init" segment (if available) the first code segment in the file.
- Grouping all "static" segments together.
- Special casing the "shift" operator to provide fast relocation of 8 and 16 bit addresses.
- Loading of segments is reduced to a few operating system calls.
- Using its own direct page which is page aligned for faster indirect access to data.
- Leaving the application open if the file contains any dynamic segments.

---

[1] If an application uses the system loader's "load_seg_by_num" call it will not work correctly without ExpressLoad installed since an additional segment has been added to the front of the file. An application should use "load_Seg_by_name" in order to load dynamic segments.

ExpressLoad Calls:

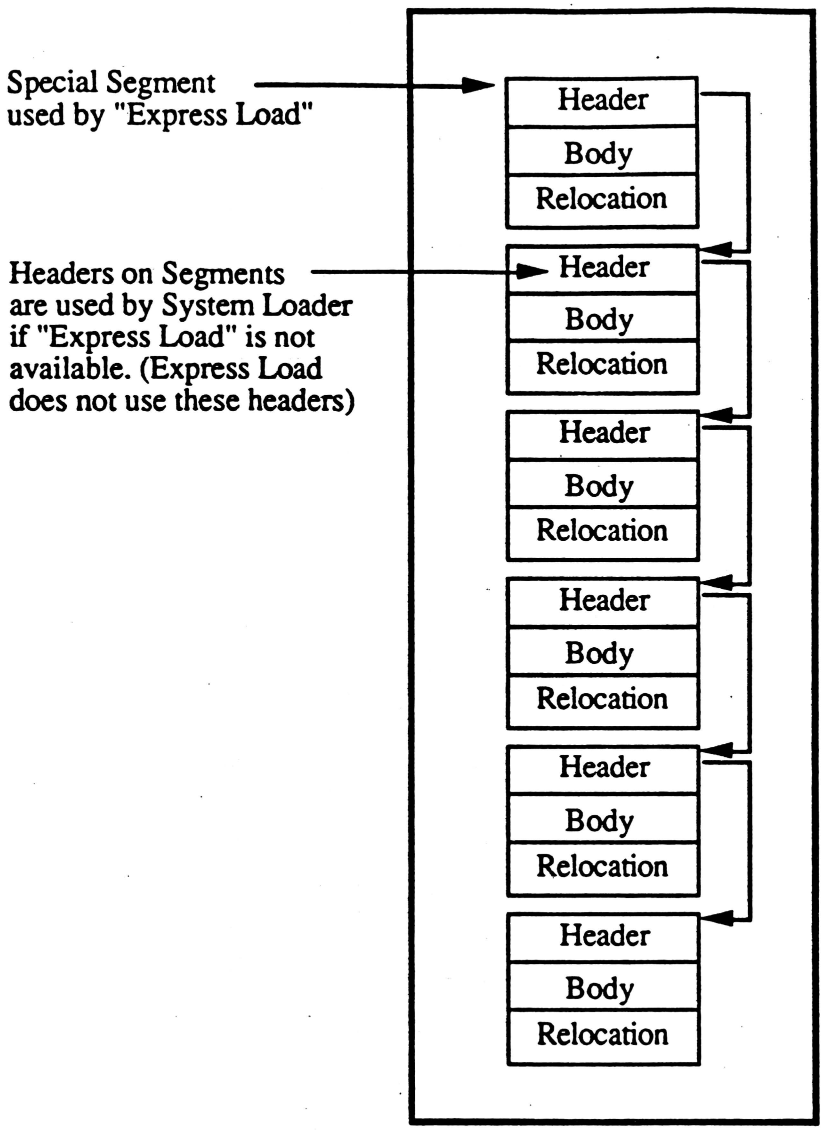
The ExpressLoad call set is the same as the standard system loader's. The exception to this rule is the Get_Load_Segment_Info. This call is not implemented by ExpressLoad since the internal data structures used by the standard system loader are not the same as the data structures used by ExpressLoad. Furthermore, ExpressLoad does not support load_from_memory operations. The load_from_memory call is passed onto the standard system loader. For a detailed description of each call, see the system loader ERS by Lou Infeld.

(1) init_load
(2) restart
(3) seg_by_num
(4) unload1
(5) seg_by_name
(6) unload2
(7) get_userID
(8) get_pathname
(9) user_shutdown
(10) initial2
(11) getpath2
(12) get_userID2

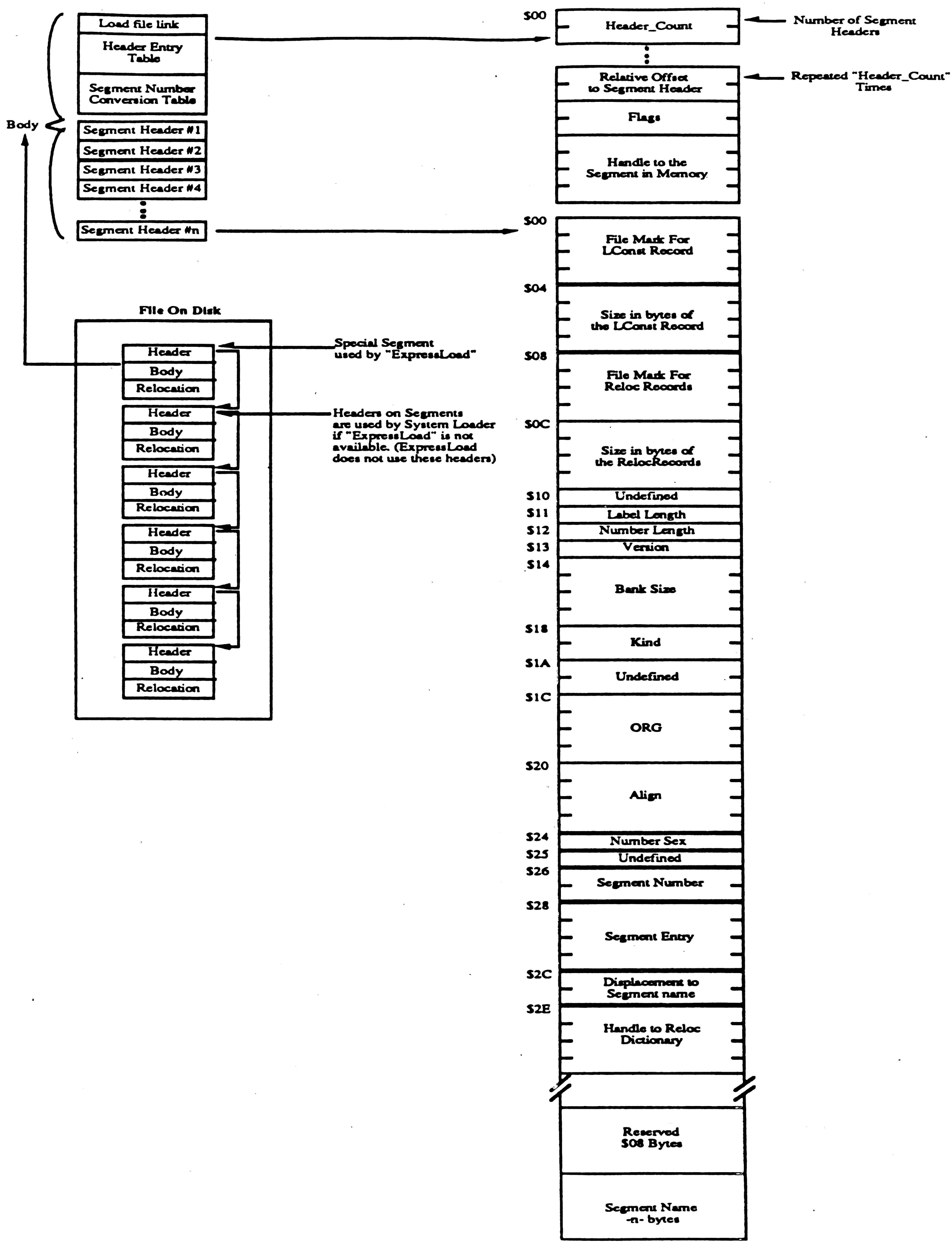
ExpressLoad file format:

The standard system loader processes files that are stored in OMF (Object Module Format: see the APW reference manuals for a detailed description of OMF). ExpressLoad processes files that are stored in OMF and contain a special segment named "ExpressLoad". Currently, the only way to build an ExpressLoad file is to use the APW tool, Express, or the MAX tool, ExpressIIGS, that will convert an OMF 2.0 file into ExpressLoad format. (Only version 2.0 of OMF can be converted to ExpressLoad format). The following diagram shows a file that is stored in ExpressLoad format.

File On Disk



ExpressLoad does not use the standard OMF headers. Instead, all the segment header information is maintained in the ExpressLoad header segment. Furthermore, ExpressLoad will keep the segment header in memory as long as the application has not been shutdown. This allows for direct access to any segments that may be needed by the application. The original segment headers are left in the file for compatibility with the standard system loader. In order to maintain compatibility with the current system loader, the ExpressLoad segment is stored as a dynamic data segment. This causes the system loader to bypass the ExpressLoad segment when doing an initial load of the application. The following diagram shows the internal format of the ExpressLoad segment.



Body
Load file link
Header Entry Table
Segment Number Conversion Table
Segment Header #1
Segment Header #2
Segment Header #3
Segment Header #4
Segment Header #n
File On Disk
Header
Body
Relocation
Special Segment used by "ExpressLoad"
Headers on Segments are used by System Loader if "ExpressLoad" is not available. (ExpressLoad does not use these headers)
$00
Header_Count
Number of Segment Headers
Relative Offset to Segment Header
Repeated "Header_Count" Times
Flags
Handle to the Segment in Memory
$00
File Mark For LConst Record
$04
Size in bytes of the LConst Record
$08
File Mark For Reloc Records
$0C
Size in bytes of the RelocRecords
$10
Undefined
$11
Label Length
$12
Number Length
$13
Version
$14
Bank Size
$18
Kind
$1A
Undefined
$1C
ORG
$20
Align
$24
Number Sex
$25
Undefined
$26
Segment Number
$28
Segment Entry
$2C
Displacement to Segment name
$2E
Handle to Reloc Dictionary
Reserved $08 Bytes
Segment Name -n- bytes

File Format in Great Detail:

The following section gives a breakdown of a file that has been converted to ExpressLoad format. The example used contains many different types of segments. In order to show a clear example of how the file has changed, both before and after dumps of the file are included in the example. The example should only be looked at if you really want to know how the ExpressLoad segment is layed out. The example is a real application that has been converted to ExpressLoad format by the APW tool Express.

Segment Header Dump of Unconverted File:

```
;
;The following dump is of the sample file before it has been converted to
;ExpressLoad format. It is listed here so the reader can refer to the dump while
;looking at the converted sample file that follows.
;

Byte count      : $00006cd5                                27861
Reserved space  : $00000000                                    0
Length          : $00005391                                21393
Kind            : $0000                      static code segment
Label length    : $00                                          0
Number length   : $04                                          4
Version         : $02                                          2
Bank size       : $00010000                                65536
Org             : $00000000                                    0
Alignment       : $00000000                                    0
Number sex      : $00                                          0
Revision        : $00                                          0
Segment number  : $0001                                        1
Entry           : $00000000                                    0
Disp to names   : $002c                                       44
Disp to data    : $003b                                       59
Load name       : ..........
Segment name    : main
```

Byte count	:	$00001a5f	6751
Reserved space	:	$00000000	0
Length	:	$00001a01	6657
Kind	:	$0010	static initialization segment
Label length	:	$00	0
Number length	:	$04	4
Version	:	$02	2
Bank size	:	$00010000	65536
Org	:	$00000000	0
Alignment	:	$00000000	0
Number sex	:	$00	0
Revision	:	$00	0
Segment number	:	$0002	2
Entry	:	$00000000	0
Disp to names	:	$002c	44
Disp to data	:	$003c	60
Load name	:	....	
Segment name	:	title	

Byte count	:	$000063a5	25509
Reserved space	:	$00000000	0
Length	:	$000051aa	20906
Kind	:	$0000	static code segment
Label length	:	$00	0
Number length	:	$04	4
Version	:	$02	2
Bank size	:	$00010000	65536
Org	:	$00000000	0
Alignment	:	$00000000	0
Number sex	:	$00	0
Revision	:	$00	0
Segment number	:	$0003	3
Entry	:	$00000000	0
Disp to names	:	$002c	44
Disp to data	:	$003a	58
Load name	:	....	
Segment name	:	shw	

```
Byte count      : $0000a85e                          43102
Reserved space  : $00000000                              0
Length          : $00009572                          38258
Kind            : $0000                static code segment
Label length    : $00                                    0
Number length   : $04                                    4
Version         : $02                                    2
Bank size       : $00010000                          65536
Org             : $00000000                              0
Alignment       : $00000000                              0
Number sex      : $00                                    0
Revision        : $00                                    0
Segment number  : $0004                                  4
Entry           : $00000000                              0
Disp to names   : $002c                                 44
Disp to data    : $003b                                 59
Load name       : ..........
Segment name    : shw2

Byte count      : $0000c038                          49208
Reserved space  : $00000000                              0
Length          : $0000a587                          42375
Kind            : $0000                static code segment
Label length    : $00                                    0
Number length   : $04                                    4
Version         : $02                                    2
Bank size       : $00010000                          65536
Org             : $00000000                              0
Alignment       : $00000000                              0
Number sex      : $00                                    0
Revision        : $00                                    0
Segment number  : $0005                                  5
Entry           : $00000000                              0
Disp to names   : $002c                                 44
Disp to data    : $003b                                 59
Load name       : ..........
Segment name    : shw3
```

Byte count	:	$0000cbd7	52183
Reserved space	:	$00000000	0
Length	:	$0000b54c	46412
Kind	:	$0000	static code segment
Label length	:	$00	0
Number length	:	$04	4
Version	:	$02	2
Bank size	:	$00010000	65536
Org	:	$00000000	0
Alignment	:	$00000000	0
Number sex	:	$00	0
Revision	:	$00	0
Segment number	:	$0006	6
Entry	:	$00000000	0
Disp to names	:	$002c	44
Disp to data	:	$003b	59
Load name	:	....	
Segment name	:	shw4	

Byte count	:	$00001fc2	8130
Reserved space	:	$00000000	0
Length	:	$00001b15	6933
Kind	:	$0000	static code segment
Label length	:	$00	0
Number length	:	$04	4
Version	:	$02	2
Bank size	:	$00010000	65536
Org	:	$00000000	0
Alignment	:	$00000000	0
Number sex	:	$00	0
Revision	:	$00	0
Segment number	:	$0007	7
Entry	:	$00000000	0
Disp to names	:	$002c	44
Disp to data	:	$003b	59
Load name	:	....	
Segment name	:	shw5	

```
Byte count      : $00009b92                    39826
Reserved space  : $00000000                        0
Length          : $00007f29                    32553
Kind            : $0000          static code segment
Label length    : $00                              0
Number length   : $04                              4
Version         : $02                              2
Bank size       : $00010000                    65536
Org             : $00000000                        0
Alignment       : $00000000                        0
Number sex      : $00                              0
Revision        : $00                              0
Segment number  : $0008                            8
Entry           : $00000000                        0
Disp to names   : $002c                           44
Disp to data    : $003b                           59
Load name       : ..........
Segment name    : shw6

Byte count      : $000027b1                    10161
Reserved space  : $00000000                        0
Length          : $000021a7                     8615
Kind            : $8000         dynamic code segment
Label length    : $00                              0
Number length   : $04                              4
Version         : $02                              2
Bank size       : $00010000                    65536
Org             : $00000000                        0
Alignment       : $00000000                        0
Number sex      : $00                              0
Revision        : $00                              0
Segment number  : $0009                            9
Entry           : $00000000                        0
Disp to names   : $002c                           44
Disp to data    : $003e                           62
Load name       : ..........
Segment name    : otherio
```

```
Byte count      : $00002c4e                          11342
Reserved space  : $00000000                              0
Length          : $00001d64                           7524
Kind            : $8000               dynamic code segment
Label length    : $00                                    0
Number length   : $04                                    4
Version         : $02                                    2
Bank size       : $00010000                          65536
Org             : $00000000                              0
Alignment       : $00000000                              0
Number sex      : $00                                    0
Revision        : $00                                    0
Segment number  : $000a                                 10
Entry           : $00000000                              0
Disp to names   : $002c                                 44
Disp to data    : $003c                                 60
Load name       : ..........
Segment name    : print

Byte count      : $0000d45a                          54362
Reserved space  : $00000000                              0
Length          : $0000b187                          45447
Kind            : $8000               dynamic code segment
Label length    : $00                                    0
Number length   : $04                                    4
Version         : $02                                    2
Bank size       : $00010000                          65536
Org             : $00000000                              0
Alignment       : $00000000                              0
Number sex      : $00                                    0
Revision        : $00                                    0
Segment number  : $000b                                 11
Entry           : $00000000                              0
Disp to names   : $002c                                 44
Disp to data    : $003e                                 62
Load name       : ..........
Segment name    : pnttool
```

Byte count	:	$00006373	25459
Reserved space	:	$00000000	0
Length	:	$0000577c	22396
Kind	:	$8000	dynamic code segment
Label length	:	$00	0
Number length	:	$04	4
Version	:	$02	2
Bank size	:	$00010000	65536
Org	:	$00000000	0
Alignment	:	$00000000	0
Number sex	:	$00	0
Revision	:	$00	0
Segment number	:	$000c	12
Entry	:	$00000000	0
Disp to names	:	$002c	44
Disp to data	:	$003f	63
Load name	:	....	
Segment name	:	pnttool2	

Byte count	:	$00004f07	20231
Reserved space	:	$00000000	0
Length	:	$00002ce4	11492
Kind	:	$8000	dynamic code segment
Label length	:	$00	0
Number length	:	$04	4
Version	:	$02	2
Bank size	:	$00010000	65536
Org	:	$00000000	0
Alignment	:	$00000000	0
Number sex	:	$00	0
Revision	:	$00	0
Segment number	:	$000d	13
Entry	:	$00000000	0
Disp to names	:	$002c	44
Disp to data	:	$003f	63
Load name	:	....	
Segment name	:	findrplc	

Byte count	: $000005e5	1509
Reserved space	: $00000000	0
Length	: $000004b1	1201
Kind	: $8000	dynamic code segment
Label length	: $00	0
Number length	: $04	4
Version	: $02	2
Bank size	: $00010000	65536
Org	: $00000000	0
Alignment	: $00000000	0
Number sex	: $00	0
Revision	: $00	0
Segment number	: $000e	14
Entry	: $00000000	0
Disp to names	: $002c	44
Disp to data	: $003f	63
Load name	:	
Segment name	: aboutshw	

Byte count	: $00001484	5252
Reserved space	: $00000000	0
Length	: $00001271	4721
Kind	: $8000	dynamic code segment
Label length	: $00	0
Number length	: $04	4
Version	: $02	2
Bank size	: $00010000	65536
Org	: $00000000	0
Alignment	: $00000000	0
Number sex	: $00	0
Revision	: $00	0
Segment number	: $000f	15
Entry	: $00000000	0
Disp to names	: $002c	44
Disp to data	: $003e	62
Load name	:	
Segment name	: convert	

```
Byte count      : $0000142a                        5162
Reserved space  : $00000000                           0
Length          : $00001077                        4215
Kind            : $8000           dynamic code segment
Label length    : $00                                 0
Number length   : $04                                 4
Version         : $02                                 2
Bank size       : $00010000                       65536
Org             : $00000000                           0
Alignment       : $00000000                           0
Number sex      : $00                                 0
Revision        : $00                                 0
Segment number  : $0010                              16
Entry           : $00000000                           0
Disp to names   : $002c                              44
Disp to data    : $003b                              59
Load name       : ..........
Segment name    : help

Byte count      : $00000ab1                        2737
Reserved space  : $00000000                           0
Length          : $00000923                        2339
Kind            : $8000           dynamic code segment
Label length    : $00                                 0
Number length   : $04                                 4
Version         : $02                                 2
Bank size       : $00010000                       65536
Org             : $00000000                           0
Alignment       : $00000000                           0
Number sex      : $00                                 0
Revision        : $00                                 0
Segment number  : $0011                              17
Entry           : $00000000                           0
Disp to names   : $002c                              44
Disp to data    : $0041                              65
Load name       : ..........
Segment name    : pagelayout
```

```
Byte count      : $0000104c                         4172
Reserved space  : $00000000                            0
Length          : $00000cd1                         3281
Kind            : $8000             dynamic code segment
Label length    : $00                                  0
Number length   : $04                                  4
Version         : $02                                  2
Bank size       : $00010000                        65536
Org             : $00000000                            0
Alignment       : $00000000                            0
Number sex      : $00                                  0
Revision        : $00                                  0
Segment number  : $0012                               18
Entry           : $00000000                            0
Disp to names   : $002c                               44
Disp to data    : $0041                               65
Load name       : ..........
Segment name    : pagenumber

Byte count      : $000007d8                         2008
Reserved space  : $00000000                            0
Length          : $000006a2                         1698
Kind            : $8000             dynamic code segment
Label length    : $00                                  0
Number length   : $04                                  4
Version         : $02                                  2
Bank size       : $00010000                        65536
Org             : $00000000                            0
Alignment       : $00000000                            0
Number sex      : $00                                  0
Revision        : $00                                  0
Segment number  : $0013                               19
Entry           : $00000000                            0
Disp to names   : $002c                               44
Disp to data    : $0040                               64
Load name       : ..........
Segment name    : pagesetup
```

```
Byte count      : $000005af                        1455
Reserved space  : $00000000                           0
Length          : $000004e2                        1250
Kind            : $8000           dynamic code segment
Label length    : $00                                 0
Number length   : $04                                 4
Version         : $02                                 2
Bank size       : $00010000                       65536
Org             : $00000000                           0
Alignment       : $00000000                           0
Number sex      : $00                                 0
Revision        : $00                                 0
Segment number  : $0014                              20
Entry           : $00000000                           0
Disp to names   : $002c                              44
Disp to data    : $003f                              63
Load name       : ..........
Segment name    : pgphlead

Byte count      : $00000e06                        3590
Reserved space  : $00000000                           0
Length          : $00000b7e                        2942
Kind            : $8000           dynamic code segment
Label length    : $00                                 0
Number length   : $04                                 4
Version         : $02                                 2
Bank size       : $00010000                       65536
Org             : $00000000                           0
Alignment       : $00000000                           0
Number sex      : $00                                 0
Revision        : $00                                 0
Segment number  : $0015                              21
Entry           : $00000000                           0
Disp to names   : $002c                              44
Disp to data    : $0041                              65
Load name       : ..........
Segment name    : setmargins
```

```
Byte count      : $00001893                                  6291
Reserved space  : $00000000                                     0
Length          : $00001620                                  5664
Kind            : $8000                      dynamic code segment
Label length    : $00                                           0
Number length   : $04                                           4
Version         : $02                                           2
Bank size       : $00010000                                 65536
Org             : $00000000                                     0
Alignment       : $00000000                                     0
Number sex      : $00                                           0
Revision        : $00                                           0
Segment number  : $0016                                        22
Entry           : $00000000                                     0
Disp to names   : $002c                                        44
Disp to data    : $003f                                        63
Load name       : ..........
Segment name    : showpage

Byte count      : $0000af05                                 44805
Reserved space  : $00000000                                     0
Length          : $0001443b                                 83003
Kind            : $4401     static private reload data segment
Label length    : $00                                           0
Number length   : $04                                           4
Version         : $02                                           2
Bank size       : $00000000                                     0
Org             : $00000000                                     0
Alignment       : $00000000                                     0
Number sex      : $00                                           0
Revision        : $00                                           0
Segment number  : $0017                                        23
Entry           : $00000000                                     0
Disp to names   : $002c                                        44
Disp to data    : $003e                                        62
Load name       : ..........
Segment name    : ~arrays
```

```
Byte count      : $00000082                                    130
Reserved space  : $00000000                                      0
Length          : $0000002e                                     46
Kind            : $4401     static private reload data segment
Label length    : $00                                            0
Number length   : $04                                            4
Version         : $02                                            2
Bank size       : $00000000                                      0
Org             : $00000000                                      0
Alignment       : $00000000                                      0
Number sex      : $00                                            0
Revision        : $00                                            0
Segment number  : $0018                                         24
Entry           : $00000000                                      0
Disp to names   : $002c                                         44
Disp to data    : $003f                                         63
Load name       : ..........
Segment name    : ~globals

Byte count      : $00000043                                     67
Reserved space  : $00000000                                      0
Length          : $00002000                                   8192
Kind            : $0012       static direct page/stack segment
Label length    : $00                                            0
Number length   : $04                                            4
Version         : $02                                            2
Bank size       : $00010000                                  65536
Org             : $00000000                                      0
Alignment       : $00000000                                      0
Number sex      : $00                                            0
Revision        : $00                                            0
Segment number  : $0019                                         25
Entry           : $00000000                                      0
Disp to names   : $002c                                         44
Disp to data    : $003d                                         61
Load name       : ..........
Segment name    : Direct
```

```
Byte count      : $00000043                                   67
Reserved space  : $00000000                                    0
Length          : $00001000                                 4096
Kind            : $0012      static direct page/stack segment
Label length    : $00                                          0
Number length   : $04                                          4
Version         : $02                                          2
Bank size       : $00010000                                65536
Org             : $00000000                                    0
Alignment       : $00000000                                    0
Number sex      : $00                                          0
Revision        : $00                                          0
Segment number  : $001a                                       26
Entry           : $00000000                                    0
Disp to names   : $002c                                       44
Disp to data    : $003d                                       61
Load name       : ..........
Segment name    : DIRECT

Byte count      : $0000040b                                 1035
Reserved space  : $00000000                                    0
Length          : $000003c4                                  964
Kind            : $0002             static jump-table segment
Label length    : $00                                          0
Number length   : $04                                          4
Version         : $02                                          2
Bank size       : $00010000                                65536
Org             : $00000000                                    0
Alignment       : $00000000                                    0
Number sex      : $00                                          0
Revision        : $00                                          0
Segment number  : $001b                                       27
Entry           : $00000000                                    0
Disp to names   : $002c                                       44
Disp to data    : $0041                                       65
Load name       : ..........
Segment name    : ~JumpTable
```

Segment Header Dump of Converted File:

```
;
;The following is a list of all the segment headers contained
;in the converted file.  Compare the order of the converted file
;to that of the unconverted file.  Note the addition of the ExpressLoad
;segment and the reordering of the segments in the file.
;
```

```
Byte count      : $00000847                          2119
Reserved space  : $00000000                             0
Length          : $000007ff                          2047
Kind            : $8001              dynamic data segment
Label length    : $00                                   0
Number length   : $04                                   4
Version         : $02                                   2
Bank size       : $00010000                         65536
Org             : $00000000                             0
Alignment       : $00000000                             0
Number sex      : $00                                   0
Revision        : $00                                   0
Segment number  : $0001                                 1
Entry           : $00000000                             0
Disp to names   : $002c                                44
Disp to data    : $0042                                66
Load name       : Rob Turner
Segment name    : ExpressLoad

Byte count      : $00001a5f                          6751
Reserved space  : $00000000                             0
Length          : $00001a01                          6657
Kind            : $0010     static initialization segment
Label length    : $00                                   0
Number length   : $04                                   4
Version         : $02                                   2
Bank size       : $00010000                         65536
Org             : $00000000                             0
Alignment       : $00000000                             0
Number sex      : $00                                   0
Revision        : $00                                   0
Segment number  : $0002                                 2
Entry           : $00000000                             0
Disp to names   : $002c                                44
Disp to data    : $003c                                60
Load name       : ..........
Segment name    : title
```

```
Byte count      : $00006cc9                          27849
Reserved space  : $00000000                              0
Length          : $00005391                          21393
Kind            : $0000                static code segment
Label length    : $00                                    0
Number length   : $04                                    4
Version         : $02                                    2
Bank size       : $00010000                          65536
Org             : $00000000                              0
Alignment       : $00000000                              0
Number sex      : $00                                    0
Revision        : $00                                    0
Segment number  : $0003                                  3
Entry           : $00000000                              0
Disp to names   : $002c                                 44
Disp to data    : $003b                                 59
Load name       : ..........
Segment name    : main

Byte count      : $00002043                           8259
Reserved space  : $00000000                              0
Length          : $00002000                           8192
Kind            : $0012     static direct page/stack segment
Label length    : $00                                    0
Number length   : $04                                    4
Version         : $02                                    2
Bank size       : $00010000                          65536
Org             : $00000000                              0
Alignment       : $00000000                              0
Number sex      : $00                                    0
Revision        : $00                                    0
Segment number  : $0004                                  4
Entry           : $00000000                              0
Disp to names   : $002c                                 44
Disp to data    : $003d                                 61
Load name       : ..........
Segment name    : Direct
```

```
Byte count      : $00001043                              4163
Reserved space  : $00000000                                 0
Length          : $00001000                              4096
Kind            : $0012        static direct page/stack segment
Label length    : $00                                       0
Number length   : $04                                       4
Version         : $02                                       2
Bank size       : $00010000                             65536
Org             : $00000000                                 0
Alignment       : $00000000                                 0
Number sex      : $00                                       0
Revision        : $00                                       0
Segment number  : $0005                                     5
Entry           : $00000000                                 0
Disp to names   : $002c                                    44
Disp to data    : $003d                                    61
Load name       : ..........
Segment name    : DIRECT

Byte count      : $000063a5                             25509
Reserved space  : $00000000                                 0
Length          : $000051aa                             20906
Kind            : $0000                   static code segment
Label length    : $00                                       0
Number length   : $04                                       4
Version         : $02                                       2
Bank size       : $00010000                             65536
Org             : $00000000                                 0
Alignment       : $00000000                                 0
Number sex      : $00                                       0
Revision        : $00                                       0
Segment number  : $0006                                     6
Entry           : $00000000                                 0
Disp to names   : $002c                                    44
Disp to data    : $003a                                    58
Load name       : ..........
Segment name    : shw
```

```
Byte count      : $0000a85e                         43102
Reserved space  : $00000000                             0
Length          : $00009572                         38258
Kind            : $0000               static code segment
Label length    : $00                                   0
Number length   : $04                                   4
Version         : $02                                   2
Bank size       : $00010000                         65536
Org             : $00000000                             0
Alignment       : $00000000                             0
Number sex      : $00                                   0
Revision        : $00                                   0
Segment number  : $0007                                 7
Entry           : $00000000                             0
Disp to names   : $002c                                44
Disp to data    : $003b                                59
Load name       : ..........
Segment name    : shw2

Byte count      : $0000c038                         49208
Reserved space  : $00000000                             0
Length          : $0000a587                         42375
Kind            : $0000               static code segment
Label length    : $00                                   0
Number length   : $04                                   4
Version         : $02                                   2
Bank size       : $00010000                         65536
Org             : $00000000                             0
Alignment       : $00000000                             0
Number sex      : $00                                   0
Revision        : $00                                   0
Segment number  : $0008                                 8
Entry           : $00000000                             0
Disp to names   : $002c                                44
Disp to data    : $003b                                59
Load name       : ..........
Segment name    : shw3
```

```
Byte count      : $0000cbd7                           52183
Reserved space  : $00000000                               0
Length          : $0000b54c                           46412
Kind            : $0000                 static code segment
Label length    : $00                                     0
Number length   : $04                                     4
Version         : $02                                     2
Bank size       : $00010000                           65536
Org             : $00000000                               0
Alignment       : $00000000                               0
Number sex      : $00                                     0
Revision        : $00                                     0
Segment number  : $0009                                   9
Entry           : $00000000                               0
Disp to names   : $002c                                  44
Disp to data    : $003b                                  59
Load name       : ..........
Segment name    : shw4

Byte count      : $00001fc2                            8130
Reserved space  : $00000000                               0
Length          : $00001b15                            6933
Kind            : $0000                 static code segment
Label length    : $00                                     0
Number length   : $04                                     4
Version         : $02                                     2
Bank size       : $00010000                           65536
Org             : $00000000                               0
Alignment       : $00000000                               0
Number sex      : $00                                     0
Revision        : $00                                     0
Segment number  : $000a                                  10
Entry           : $00000000                               0
Disp to names   : $002c                                  44
Disp to data    : $003b                                  59
Load name       : ..........
Segment name    : shw5
```

```
Byte count      : $00009b92                              39826
Reserved space  : $00000000                                  0
Length          : $00007f29                              32553
Kind            : $0000                    static code segment
Label length    : $00                                        0
Number length   : $04                                        4
Version         : $02                                        2
Bank size       : $00010000                              65536
Org             : $00000000                                  0
Alignment       : $00000000                                  0
Number sex      : $00                                        0
Revision        : $00                                        0
Segment number  : $000b                                     11
Entry           : $00000000                                  0
Disp to names   : $002c                                     44
Disp to data    : $003b                                     59
Load name       : ..........
Segment name    : shw6
```

```
Byte count      : $000162bb                              90811
Reserved space  : $00000000                                  0
Length          : $0001443b                              83003
Kind            : $4401     static private reload data segment
Label length    : $00                                        0
Number length   : $04                                        4
Version         : $02                                        2
Bank size       : $00000000                                  0
Org             : $00000000                                  0
Alignment       : $00000000                                  0
Number sex      : $00                                        0
Revision        : $00                                        0
Segment number  : $000c                                     12
Entry           : $00000000                                  0
Disp to names   : $002c                                     44
Disp to data    : $003e                                     62
Load name       : ..........
Segment name    : ~arrays
```

```
Byte count      : $00000082                             130
Reserved space  : $00000000                               0
Length          : $0000002e                              46
Kind            : $4401     static private reload data segment
Label length    : $00                                     0
Number length   : $04                                     4
Version         : $02                                     2
Bank size       : $00000000                               0
Org             : $00000000                               0
Alignment       : $00000000                               0
Number sex      : $00                                     0
Revision        : $00                                     0
Segment number  : $000d                                  13
Entry           : $00000000                               0
Disp to names   : $002c                                  44
Disp to data    : $003f                                  63
Load name       : ..........
Segment name    : ~globals


Byte count      : $0000040b                            1035
Reserved space  : $00000000                               0
Length          : $000003c4                             964
Kind            : $0002              static jump-table segment
Label length    : $00                                     0
Number length   : $04                                     4
Version         : $02                                     2
Bank size       : $00010000                           65536
Org             : $00000000                               0
Alignment       : $00000000                               0
Number sex      : $00                                     0
Revision        : $00                                     0
Segment number  : $000e                                  14
Entry           : $00000000                               0
Disp to names   : $002c                                  44
Disp to data    : $0041                                  65
Load name       : ..........
Segment name    : ~JumpTable
```

```
Byte count      : $000027b1                      10161
Reserved space  : $00000000                          0
Length          : $000021a7                       8615
Kind            : $8000          dynamic code segment
Label length    : $00                                0
Number length   : $04                                4
Version         : $02                                2
Bank size       : $00010000                      65536
Org             : $00000000                          0
Alignment       : $00000000                          0
Number sex      : $00                                0
Revision        : $00                                0
Segment number  : $000f                             15
Entry           : $00000000                          0
Disp to names   : $002c                             44
Disp to data    : $003e                             62
Load name       : ..........
Segment name    : otherio

Byte count      : $00002c4e                      11342
Reserved space  : $00000000                          0
Length          : $00001d64                       7524
Kind            : $8000          dynamic code segment
Label length    : $00                                0
Number length   : $04                                4
Version         : $02                                2
Bank size       : $00010000                      65536
Org             : $00000000                          0
Alignment       : $00000000                          0
Number sex      : $00                                0
Revision        : $00                                0
Segment number  : $0010                             16
Entry           : $00000000                          0
Disp to names   : $002c                             44
Disp to data    : $003c                             60
Load name       : ..........
Segment name    : print
```

```
Byte count      : $0000d45a                         54362
Reserved space  : $00000000                             0
Length          : $0000b187                         45447
Kind            : $8000              dynamic code segment
Label length    : $00                                   0
Number length   : $04                                   4
Version         : $02                                   2
Bank size       : $00010000                         65536
Org             : $00000000                             0
Alignment       : $00000000                             0
Number sex      : $00                                   0
Revision        : $00                                   0
Segment number  : $0011                                17
Entry           : $00000000                             0
Disp to names   : $002c                                44
Disp to data    : $003e                                62
Load name       : ..........
Segment name    : pnttool

Byte count      : $00006373                         25459
Reserved space  : $00000000                             0
Length          : $0000577c                         22396
Kind            : $8000              dynamic code segment
Label length    : $00                                   0
Number length   : $04                                   4
Version         : $02                                   2
Bank size       : $00010000                         65536
Org             : $00000000                             0
Alignment       : $00000000                             0
Number sex      : $00                                   0
Revision        : $00                                   0
Segment number  : $0012                                18
Entry           : $00000000                             0
Disp to names   : $002c                                44
Disp to data    : $003f                                63
Load name       : ..........
Segment name    : pnttool2
```

Byte count	:	$00004f07	20231
Reserved space	:	$00000000	0
Length	:	$00002ce4	11492
Kind	:	$8000	dynamic code segment
Label length	:	$00	0
Number length	:	$04	4
Version	:	$02	2
Bank size	:	$00010000	65536
Org	:	$00000000	0
Alignment	:	$00000000	0
Number sex	:	$00	0
Revision	:	$00	0
Segment number	:	$0013	19
Entry	:	$00000000	0
Disp to names	:	$002c	44
Disp to data	:	$003f	63
Load name	:	....	
Segment name	:	findrplc	

Byte count	:	$000005e5	1509
Reserved space	:	$00000000	0
Length	:	$000004b1	1201
Kind	:	$8000	dynamic code segment
Label length	:	$00	0
Number length	:	$04	4
Version	:	$02	2
Bank size	:	$00010000	65536
Org	:	$00000000	0
Alignment	:	$00000000	0
Number sex	:	$00	0
Revision	:	$00	0
Segment number	:	$0014	20
Entry	:	$00000000	0
Disp to names	:	$002c	44
Disp to data	:	$003f	63
Load name	:	....	
Segment name	:	aboutshw	

```
Byte count      : $00001484                      5252
Reserved space  : $00000000                         0
Length          : $00001271                      4721
Kind            : $8000          dynamic code segment
Label length    : $00                               0
Number length   : $04                               4
Version         : $02                               2
Bank size       : $00010000                     65536
Org             : $00000000                         0
Alignment       : $00000000                         0
Number sex      : $00                               0
Revision        : $00                               0
Segment number  : $0015                            21
Entry           : $00000000                         0
Disp to names   : $002c                            44
Disp to data    : $003e                            62
Load name       : ..........
Segment name    : convert

Byte count      : $0000142a                      5162
Reserved space  : $00000000                         0
Length          : $00001077                      4215
Kind            : $8000          dynamic code segment
Label length    : $00                               0
Number length   : $04                               4
Version         : $02                               2
Bank size       : $00010000                     65536
Org             : $00000000                         0
Alignment       : $00000000                         0
Number sex      : $00                               0
Revision        : $00                               0
Segment number  : $0016                            22
Entry           : $00000000                         0
Disp to names   : $002c                            44
Disp to data    : $003b                            59
Load name       : ..........
Segment name    : help
```

```
Byte count      : $00000ab1                          2737
Reserved space  : $00000000                             0
Length          : $00000923                          2339
Kind            : $8000              dynamic code segment
Label length    : $00                                   0
Number length   : $04                                   4
Version         : $02                                   2
Bank size       : $00010000                         65536
Org             : $00000000                             0
Alignment       : $00000000                             0
Number sex      : $00                                   0
Revision        : $00                                   0
Segment number  : $0017                                23
Entry           : $00000000                             0
Disp to names   : $002c                                44
Disp to data    : $0041                                65
Load name       : ..........
Segment name    : pagelayout

Byte count      : $0000104c                          4172
Reserved space  : $00000000                             0
Length          : $00000cd1                          3281
Kind            : $8000              dynamic code segment
Label length    : $00                                   0
Number length   : $04                                   4
Version         : $02                                   2
Bank size       : $00010000                         65536
Org             : $00000000                             0
Alignment       : $00000000                             0
Number sex      : $00                                   0
Revision        : $00                                   0
Segment number  : $0018                                24
Entry           : $00000000                             0
Disp to names   : $002c                                44
Disp to data    : $0041                                65
Load name       : ..........
Segment name    : pagenumber
```

Byte count	:	$000007d8	2008
Reserved space	:	$00000000	0
Length	:	$000006a2	1698
Kind	:	$8000	dynamic code segment
Label length	:	$00	0
Number length	:	$04	4
Version	:	$02	2
Bank size	:	$00010000	65536
Org	:	$00000000	0
Alignment	:	$00000000	0
Number sex	:	$00	0
Revision	:	$00	0
Segment number	:	$0019	25
Entry	:	$00000000	0
Disp to names	:	$002c	44
Disp to data	:	$0040	64
Load name	:	....	
Segment name	:	pagesetup	

Byte count	:	$000005af	1455
Reserved space	:	$00000000	0
Length	:	$000004e2	1250
Kind	:	$8000	dynamic code segment
Label length	:	$00	0
Number length	:	$04	4
Version	:	$02	2
Bank size	:	$00010000	65536
Org	:	$00000000	0
Alignment	:	$00000000	0
Number sex	:	$00	0
Revision	:	$00	0
Segment number	:	$001a	26
Entry	:	$00000000	0
Disp to names	:	$002c	44
Disp to data	:	$003f	63
Load name	:	....	
Segment name	:	pgphlead	

```
Byte count      : $00000e06                                 3590
Reserved space  : $00000000                                    0
Length          : $00000b7e                                 2942
Kind            : $8000                    dynamic code segment
Label length    : $00                                          0
Number length   : $04                                          4
Version         : $02                                          2
Bank size       : $00010000                                65536
Org             : $00000000                                    0
Alignment       : $00000000                                    0
Number sex      : $00                                          0
Revision        : $00                                          0
Segment number  : $001b                                       27
Entry           : $00000000                                    0
Disp to names   : $002c                                       44
Disp to data    : $0041                                       65
Load name       : ..........
Segment name    : setmargins

Byte count      : $00001893                                 6291
Reserved space  : $00000000                                    0
Length          : $00001620                                 5664
Kind            : $8000                    dynamic code segment
Label length    : $00                                          0
Number length   : $04                                          4
Version         : $02                                          2
Bank size       : $00010000                                65536
Org             : $00000000                                    0
Alignment       : $00000000                                    0
Number sex      : $00                                          0
Revision        : $00                                          0
Segment number  : $001c                                       28
Entry           : $00000000                                    0
Disp to names   : $002c                                       44
Disp to data    : $003f                                       63
Load name       : ..........
Segment name    : showpage
```

Hex Dump of ExpressLoad Segment in Converted File:

```
;
;Below is a dump of the ExpressLoad segment for the sample file.
;What follows the dumpobj is a detailed description of the data
;contained in the LConst portion of the segment.  The following
;dump should be used as a reference while examining the detailed
;description of the LConst portion of the ExpressLoad segment.
;

Byte count      : $00000847                                 2119
Reserved space  : $00000000                                    0
Length          : $000007ff                                 2047
Kind            : $8001                    dynamic data segment
Label length    : $00                                          0
Number length   : $04                                          4
Version         : $02                                          2
Bank size       : $00010000                                65536
Org             : $00000000                                    0
Alignment       : $00000000                                    0
Number sex      : $00                                          0
Revision        : $00                                          0
Segment number  : $0001                                        1
Entry           : $00000000                                    0
Disp to names   : $002c                                       44
Disp to data    : $0042                                       66
Load name       : Rob Turner
Segment name    : ExpressLoad

000042 000000 | LCONST     ($f2) | $000007ff
000047 000000 | 00000000 1a000e01  00000000 00004601 | ..............F.
000057 000010 | 00000000 00007d01  00000000 0000b601 | ......}.........
000067 000020 | 00000000 0000ef01  00000000 00002502 | ..............%.
000077 000030 | 00000000 00005c02  00000000 00009302 | ......\.........
000087 000040 | 00000000 0000ca02  00000000 00000103 | ................
000097 000050 | 00000000 00003803  00000000 00007203 | ......8.......r.
0000a7 000060 | 00000000 0000ad03  00000000 0000ea03 | ................
0000b7 000070 | 00000000 00002404  00000000 00005c04 | ......$.......\.
0000c7 000080 | 00000000 00009604  00000000 0000d104 | ................
0000d7 000090 | 00000000 00000c05  00000000 00004705 | ..............G.
0000e7 0000a0 | 00000000 00008105  00000000 0000b805 | ................
0000f7 0000b0 | 00000000 0000f505  00000000 00003206 | ..............2.
000107 0000c0 | 00000000 00006e06  00000000 0000a906 | ......n.........
000117 0000d0 | 00000000 0000e606  00000000 00000300 | ................
000127 0000e0 | 02000600 07000800  09000a00 0b000f00 | ................
000137 0000f0 | 10001100 12001300  14001500 16001700 | ................
000147 000100 | 18001900 1a001b00  1c000c00 0d000400 | ................
000157 000110 | 05000e00 88080000  011a0000 89220000 | ............."..
000167 000120 | 1c000000 00000402  00000100 10000000 | ................
000177 000130 | 00000000 00000000  00000200 00000000 | ................
000187 000140 | 2c003c00 00000000  00000000 00000574 | ,.<............t
000197 000150 | 69746c65 e6220000  91530000 77760000 | itle."...S..wv..
0001a7 000160 | f7180000 00000402  00000100 00000000 | ................
0001b7 000170 | 00000000 00000000  00000300 00000000 | ................
```

```
0001c7 000180 | 2c003b00 00000000 00000000 0000046d | ,.;............m
0001d7 000190 | 61696eb1 8f000000 20000000 00000000 | ain..... .......
0001e7 0001a0 | 00000000 00040200 00010012 00000000 | ................
0001f7 0001b0 | 00000000 00000000 00040000 0000002c | ...............,
000207 0001c0 | 003d0000 00000000 00000000 00064469 | .=............Di
000217 0001d0 | 72656374 f4af0000 00100000 00000000 | rect............
000227 0001e0 | 00000000 00000402 00000100 12000000 | ................
000237 0001f0 | 00000000 00000000 00000500 00000000 | ................
000247 000200 | 2c003d00 00000000 00000000 00000644 | ,.=............D
000257 000210 | 49524543 5434c000 00aa5100 00de1101 | IRECT4....Q.....
000267 000220 | 00bb1100 00000004 02000001 00000000 | ................
000277 000230 | 00000000 00000000 00000006 00000000 | ................
000287 000240 | 002c003a 00000000 00000000 00000003 | .,.:............
000297 000250 | 6c656fda 23010072 9500004c b90100ab | shw.#..r...L....
0002a7 000260 | 12000000 00040200 00010000 00000000 | ................
0002b7 000270 | 00000000 00000000 00070000 0000002c | ...............,
0002c7 000280 | 003b0000 00000000 00000000 00046c65 | .;............sh
0002d7 000290 | 6f3238cc 010087a5 0000bf71 0200701a | w28........q..p.
0002e7 0002a0 | 00000000 04020000 01000000 00000000 | ................
0002f7 0002b0 | 00000000 00000000 08000000 00002c00 | ..............,.
000307 0002c0 | 3b000000 00000000 00000000 046c656f | ;............shw
000317 0002d0 | 33708c02 004cb500 00bc4103 004a1600 | 3p...L....A..J..
000327 0002e0 | 00000004 02000001 00000000 00000000 | ................
000337 0002f0 | 00000000 00000009 00000000 002c003b | .............,.;
000347 000300 | 00000000 00000000 00000004 6c656f34 | ............shw4
000357 000310 | 47580300 151b0000 5c730300 6c040000 | GX......\s..l...
000367 000320 | 00000402 00000100 00000000 00000000 | ................
000377 000330 | 00000000 00000a00 00000000 2c003b00 | ............,.;.
000387 000340 | 00000000 00000000 0000046c 656f3509 | ...........shw5.
000397 000350 | 78030029 7f000032 f7030028 1c000000 | x..)...2...(....
0003a7 000360 | 00040200 00010000 00000000 00000000 | ................
0003b7 000370 | 00000000 000b0000 0000002c 003b0000 | ...........,.;..
0003c7 000380 | 00000000 00000000 00046c65 6f369e13 | ..........shw6..
0003d7 000390 | 04003b44 0100d957 05003c1e 00000000 | ..;D...W..<.....
0003e7 0003a0 | 04020000 00000144 00000000 00000000 | .......D........
0003f7 0003b0 | 00000000 0c000000 00002c00 3e000000 | ..........,.>...
000407 0003c0 | 00000000 00000000 077e6172 72617973 | .........~arrays
000417 0003d0 | 5a760500 2e000000 88760500 0f000000 | Zv.......v......
000427 0003e0 | 00000402 00000000 01440000 00000000 | .........D......
000437 0003f0 | 00000000 00000d00 00000000 2c003f00 | ............,.?.
000447 000400 | 00000000 00000000 0000087e 676c6f62 | ...........~glob
000457 000410 | 616c73de 760500c4 03000000 00000000 | als.v...........
000467 000420 | 00000000 00040200 00010002 00000000 | ................
000477 000430 | 00000000 00000000 000e0000 0000002c | ...............,
000487 000440 | 00410000 00000000 00000000 000a7e4a | .A............~J
000497 000450 | 756d7054 61626c65 e67a0500 a7210000 | umpTable.z...!..
0004a7 000460 | 8d9c0500 c6050000 00000402 00000100 | ................
0004b7 000470 | 00800000 00000000 00000000 00000f00 | ................
0004c7 000480 | 00000000 2c003e00 00000000 00000000 | ....,.>.........
0004d7 000490 | 0000076f 74686572 696f95a2 0500641d | ...otherio....d.
0004e7 0004a0 | 0000f9bf 0500a80e 00000000 04020000 | ................
0004f7 0004b0 | 01000080 00000000 00000000 00000000 | ................
000507 0004c0 | 10000000 00002c00 3c000000 00000000 | ......,.<.......
000517 0004d0 | 00000000 05707269 6e74e5ce 050087b1 | .....print......
```

```
000527 0004e0 | 00006c80 06008f22  00000000 04020000 | ..l....".......
000537 0004f0 | 01000080 00000000  00000000 00000000 | ................
000547 000500 | 11000000 00002c00  3e000000 00000000 | ......,.>.......
000557 000510 | 00000000 07617274  746f6f6c 40a30600 | .....pnttool@...
000567 000520 | 7c570000 bcfa0600  b20b0000 00000402 | |W..............
000577 000530 | 00000100 00800000  00000000 00000000 | ................
000587 000540 | 00001200 00000000  2c003f00 00000000 | ........,.?.....
000597 000550 | 00000000 00000861  7274746f 6f6c32b3 | .......pnttool2.
0005a7 000560 | 060700e4 2c000097  330700de 21000000 | ....,...3...!...
0005b7 000570 | 00040200 00010000  80000000 00000000 | ................
0005c7 000580 | 00000000 00130000  0000002c 003f0000 | ...........,.?..
0005d7 000590 | 00000000 00000000  00086669 6e647270 | ..........findrp
0005e7 0005a0 | 6c63ba55 0700b104  00006b5a 0700ef00 | lc.U......kZ....
0005f7 0005b0 | 00000000 04020000  01000080 00000000 | ................
000607 0005c0 | 00000000 00000000  14000000 00002c00 | ..............,.
000617 0005d0 | 3f000000 00000000  00000000 0861626f | ?............abo
000627 0005e0 | 75746c65 6f9e5b07  00711200 000f6e07 | utshw.[..q....n.
000637 0005f0 | 00cf0100 00000004  02000001 00008000 | ................
000647 000600 | 00000000 00000000  00000015 00000000 | ................
000657 000610 | 002c003e 00000000  00000000 00000007 | .,.>............
000667 000620 | 636f6e76 6572741f  70070077 10000096 | convert.p..w....
000677 000630 | 80070072 03000000  00040200 00010000 | ...r............
000687 000640 | 80000000 00000000  00000000 00160000 | ................
000697 000650 | 0000002c 003b0000  00000000 00000000 | ...,.;..........
0006a7 000660 | 00046865 6c704f84  07002309 0000728d | ..helpO...#...r.
0006b7 000670 | 07004701 00000000  04020000 01000080 | ..G.............
0006c7 000680 | 00000000 00000000  00000000 17000000 | ................
0006d7 000690 | 00002c00 41000000  00000000 00000000 | ..,.A...........
0006e7 0006a0 | 0a706167 656c6179  6f757400 8f0700d1 | .pagelayout.....
0006f7 0006b0 | 0c0000d1 9b070034  03000000 00040200 | .......4........
000707 0006c0 | 00010000 80000000  00000000 00000000 | ................
000717 0006d0 | 00180000 0000002c  00410000 00000000 | .......,.A......
000727 0006e0 | 00000000 000a7061  67656e75 6d626572 | ......pagenumber
000737 0006f0 | 4b9f0700 a2060000  eda50700 f0000000 | K...............
000747 000700 | 00000402 00000100  00800000 00000000 | ................
000757 000710 | 00000000 00001900  00000000 2c004000 | ............,.@.
000767 000720 | 00000000 00000000  00000970 61676573 | ...........pages
000777 000730 | 65747570 22a70700  e2040000 04ac0700 | etup"...........
000787 000740 | 88000000 00000402  00000100 00800000 | ................
000797 000750 | 00000000 00000000  00001a00 00000000 | ................
0007a7 000760 | 2c003f00 00000000  00000000 00000870 | ,.?............p
0007b7 000770 | 6770686c 656164d3  ac07007e 0b000051 | gphlead....~...Q
0007c7 000780 | b8070041 02000000  00040200 00010000 | ...A............
0007d7 000790 | 80000000 00000000  00000000 001b0000 | ................
0007e7 0007a0 | 0000002c 00410000  00000000 00000000 | ...,.A..........
0007f7 0007b0 | 000a7365 746d6172  67696e73 d7ba0700 | ..setmargins....
000807 0007c0 | 20160000 f7d00700  2e020000 00000402 |  ...............
000817 0007d0 | 00000100 00800000  00000000 00000000 | ................
000827 0007e0 | 00001c00 00000000  2c003f00 00000000 | ........,.?.....
000837 0007f0 | 00000000 00000873  686f7770 616765   | .......showpage
000846 0007ff | END          ($00) |
```

Description of ExpressLoad Segment's LConst Data:

```
;
;
;Below is a break down of the ExpressLoad segment.  The break down
;is from the above dumpobj of the sample file.
;
;

Byte count      : $00000847                                   2119
Reserved space  : $00000000                                      0
Length          : $000007ff                                   2047
Kind            : $8001                       dynamic data segment
Label length    : $00                                            0
Number length   : $04                                            4
Version         : $02                                            2
Bank size       : $00010000                                  65536
Org             : $00000000                                      0
Alignment       : $00000000                                      0
Number sex      : $00                                            0
Revision        : $00                                            0
Segment number  : $0001                                          1
Entry           : $00000000                                      0
Disp to names   : $002c                                         44
Disp to data    : $0042                                         66
Load name       : Rob Turner
Segment name    : ExpressLoad

000042 000000 | LCONST     ($f2) | $000007ff

000047 000000   00000000        ;Reserved Must be set to zero
;
;Note:     ALL Words and Longs are stored in 65816 order, i.e. low byte
;          first.
;
;
;The first word after the reserved field contains the number
;of segments in the ExpressHeader Segment minus 1.
;
                1a00            ;Number of segments in ExpressSeg.

;
;To calculate the address of the segment header, add the
;relative address to the current address.  For example, the
;following segment header can be found at:
;
;    Relative Address ($010E) + Current Address ($0006) = $0114
;
; The above example shows that the first segment header should be
; at address $0114 in this segment.
;
00004D 000006   0e01            ;Relative Offset to the first header.
                0000            ;Flags word. Must be set to zero.
                00000000        ;Holds Handle. Must be set to zero.
```

```
                4601        ;Relative offset to the second header.
000057 000010   0000        ;Flags word. Must be set to zero.
                00000000    ;Holds Handle. Must be set to zero.

                7d01        ;Relative offset to the second header.
                0000        ;Flags word. Must be set to zero.
                00000000    ;Holds Handle. Must be set to zero.

                b601        ;Relative offset to segment header.
000067 000020   0000        ;Flags word. Must be set to zero.
                00000000    ;Holds Handle. Must be set to zero.

                ef01        ;Relative offset to segment header.
                0000        ;Flags word. Must be set to zero.
                00000000    ;Holds Handle. Must be set to zero.

                2502        ;Relative offset to segment header.
000077 000030   0000        ;Flags word. Must be set to zero.
                00000000    ;Holds Handle. Must be set to zero.

                5c02        ;Relative offset to segment header.
                0000        ;Flags word. Must be set to zero.
                00000000    ;Holds Handle. Must be set to zero.

                9302        ;Relative offset to segment header.
000087 000040   0000        ;Flags word. Must be set to zero.
                00000000    ;Holds Handle. Must be set to zero.

                ca02        ;Relative offset to segment header.
                0000        ;Flags word. Must be set to zero.
                00000000    ;Holds Handle. Must be set to zero.

                0103        ;Relative offset to segment header.
000097 000050   0000        ;Flags word. Must be set to zero.
                00000000    ;Holds Handle. Must be set to zero.

                3803        ;Relative offset to segment header.
                0000        ;Flags word. Must be set to zero.
                00000000    ;Holds Handle. Must be set to zero.

                7203        ;Relative offset to segment header.
0000a7 000060   0000        ;Flags word. Must be set to zero.
                00000000    ;Holds Handle. Must be set to zero.

                ad03        ;Relative offset to segment header.
                0000        ;Flags word. Must be set to zero.
                00000000    ;Holds Handle. Must be set to zero.

                ea03        ;Relative offset to segment header.
0000b7 000070   0000        ;Flags word. Must be set to zero.
                00000000    ;Holds Handle. Must be set to zero.
```

```
                2404             ;Relative offset to segment header.
                0000             ;Flags word. Must be set to zero.
                00000000         ;Holds Handle. Must be set to zero.

                5c04             ;Relative offset to segment header.
0000c7 000080   0000             ;Flags word. Must be set to zero.
                00000000         ;Holds Handle. Must be set to zero.

                9604             ;Relative offset to segment header.
                0000             ;Flags word. Must be set to zero.
                00000000         ;Holds Handle. Must be set to zero.

                d104             ;Relative offset to segment header.
0000d7 000090   0000             ;Flags word. Must be set to zero.
                00000000         ;Holds Handle. Must be set to zero.

                0c05             ;Relative offset to segment header.
                0000             ;Flags word. Must be set to zero.
                00000000         ;Holds Handle. Must be set to zero.

                4705             ;Relative offset to segment header.
0000e7 0000a0   0000             ;Flags word. Must be set to zero.
                00000000         ;Holds Handle. Must be set to zero.

                8105             ;Relative offset to segment header.
                0000             ;Flags word. Must be set to zero.
                00000000         ;Holds Handle. Must be set to zero.

                b805             ;Relative offset to segment header.
0000f7 0000b0   0000             ;Flags word. Must be set to zero.
                00000000         ;Holds Handle. Must be set to zero.

                f505             ;Relative offset to segment header.
                0000             ;Flags word. Must be set to zero.
                00000000         ;Holds Handle. Must be set to zero.

                3206             ;Relative offset to segment header.
000107 0000c0   0000             ;Flags word. Must be set to zero.
                00000000         ;Holds Handle. Must be set to zero.

                6e06             ;Relative offset to segment header.
                0000             ;Flags word. Must be set to zero.
                00000000         ;Holds Handle. Must be set to zero.

                a906             ;Relative offset to segment header.
000117 0000d0   0000             ;Flags word. Must be set to zero.
                00000000         ;Holds Handle. Must be set to zero.

                e606             ;Relative offset to segment header.
                0000             ;Flags word. Must be set to zero.
                00000000         ;Holds Handle. Must be set to zero.
```

```
;
;Now we begin the segment number conversion table.
;The segment number conversion table is used to gain
;compatibility with the "load_segment_by_number" call.
;
;If the caller wishes to load segment number 4 from
;the original file ExpressLoad will convert the number via the
;conversion table to the current segment number.
;
;IMPORTANT:     An application should never use the
;               load_seg_by_num call.  The application
;               should use the load_seg_by_name
;               call which will make the file compatible
;               with both ExpressLoad and the system loader.
;
                0300            ;original seg $01 is now seg $03
000127 0000e0   0200            ;original seg $02 is now seg $02
                0600            ;original seg $03 is now seg $06
                0700
                0800
                0900
                0a00
                0b00
                0f00
000137 0000f0   1000            ;original seg $0A is now seg $10
                1100
                1200
                1300
                1400
                1500
                1600
                1700
000147 000100   1800
                1900
                1a00
                1b00
                1c00
                0c00
                0d00
                0400
000157 000110   0500
                0e00
;
```

```
;Below is the beginning of the first segment header.
;
;
;Address of segment header in file is $000114
;
                  88080000        ;File mark of segments LConst.
                  011a0000        ;Number of bytes in the LConst.
                  89220000        ;File mark of relocation dictionary.
000167 000120     1c000000        ;number of bytes in the dictionary.
                  00              ;Reserved must be zero.
                  00              ;Label length.
                  04              ;number size.
                  02              ;OMF version.
                  00000100        ;Bank Size = $10000
                  1000            ;Segment kind field.
                  0000            ;reserved must be zero.
000177 000130     00000000        ;ORG
                  00000000        ;Align
                  00              ;Number Sex
                  00              ;reserved must zero.
                  0200            ;Segment number in file.
                  00000000        ;Segment Entry.
000187 000140     2c00            ;Displacement to segment name
                  3c000000        ;undefined/placeholder for memory data
                  00000000        ;Reserved 8 bytes total
                  00000000
                  05              ;Length of the segment name
                  74              ;Segment name "title"
000197 000150     69746c65

;
;Below is the beginning of the second segment header.
;
                  e6220000        ;File mark of segments LConst.
                  91530000        ;Number of bytes in the LConst.
                  77760000        ;File mark of relocation dictionary.
0001a7 000160     f7180000        ;number of bytes in the dictionary.
                  00              ;Reserved must be zero.
                  00              ;Label length.
                  04              ;number size.
                  02              ;OMF version.
                  00000100        ;Bank Size = $10000
                  0000            ;Segment kind field.
                  0000            ;reserved must be zero.
0001b7 000170     00000000        ;ORG
                  00000000        ;Align
                  00              ;Number Sex
                  00              ;reserved must zero.
                  0300            ;Segment number in file.
                  00000000        ;Segment Entry.
0001c7 000180     2c00            ;Displacement to segment name
                  3b000000        ;undefined/placeholder for memory data
                  00000000        ;Reserved 8 bytes total
                  00000000
```

```
                  04               ;Length of the segment name
                  6d               ;Segment name "main"
0001d7 000190     61696e           ;rest of segment name
;
;Below is the remaining segment names.  The expanded code
;is not listed here due to space considerations.
;

Direct
DIRECT
shw
shw2
shw3
shw4
shw5
shw6
~arrays
~globals
~JumpTable
otherio
print
pnttool
pnttool2
findrplc
aboutshw
convert
help
pagelayout
pagenumber
pagesetup
pgphlead
setmargins
showpage

000846 0007ff     END          ($00) |
```

GLoader/GQuit Delta ERS

External

Version 0.08

by Cheryl Ewy

Revision History

12-07-88	0.01	Initial release
12-09-88	0.02	Express Loader support added
		Custom boot flag bits changed and new bit added
01-06-89	0.03	GS.OS.DEV support added
01-11-89	0.04	I/O redirection support added
01-24-89	0.05	New Express Loader information added
		Splash screen changes added
		Init file changes added
02-09-89	0.06	Reserved bank $01 space added
		.CONSOLE check added
03-10-89	0.07	512K system support added
		FST installation changes
		ProDOS 8 compatibility changes
		OS switching changes
		Boot file changes
03-22-89	0.08	More OS switching changes
		"GS/OS aware" support added
		Cache Manager support added
		Slot Arbiter support added

Introduction

This document describes the changes being made to GLoader and GQuit for System Disk 5.0.

Resource Manager Support

GLoader now attempts to load and execute the */SYSTEM/SYSTEM.SETUP/RESOURCE.MGR file immediately after executing the TOOL.SETUP file. The RESOURCE.MGR file must have a filetype of $B6. If the file does not exist or if it has the wrong filetype, a fatal error will be reported.

Init File Changes

When GLoader loads and executes an init file (from the SYSTEM.SETUP directory) it now pushes a word of 0 on the stack just before control is passed to the init file. If a permanent init file (filetype $B6) decides that it wants to be shutdown, it should set the low bit of the word on the stack. The init file will then be shutdown when control returns to GLoader. Temporary init files (filetype $B7) are always shutdown after they have executed.

Second Entry Point Added to GLoader

A second entry point has been added to GLoader at location $006803. This entry point is used by the AppleShare boot code and can be used by any boot file which needs it. Entry conditions are the same as for the original entry point at $006800 with the additional requirements that the X and Y registers contain 0 and that the A register contain "custom boot flags" which are used by GLoader to customize the boot process. This was required in order to be able to boot from AppleShare. The "custom boot flags" are defined as follows -

bit 15 - memory manager/tool dispatcher flag
bit 14 - interrups/SCC flag
bit 13 - OS_switch flag
bit 12 - boot driver flag
bit 11 - priority vector flag
bits 10 through 0 - reserved (must be set to 0)

If the A register contains 0, the boot process will be the same as if the original entry point was used.

If bit 15 is set to 1, the file system's boot code must initialize the memory manager/tool dispatcher itself. GLoader normally does this as part of the boot process but will not do it if the bit flag is set. This is useful if the file system

needs to allocate memory other than what is saved by GLoader for operating system switches.

If bit 14 is set to 1, the file system's boot code should have cleared interrupts and reset the SCC. GLoader normally does this as part of the boot process but will not do it if the bit flag is set.

If bit 13 is set to 1, the file system's boot code should have set the OS_switch vector. GLoader normally does this as part of the boot process but will not do it if the bit flag is set.

If bit 12 is set to 1, the start FST requires a custom boot driver to be loaded before the start FST is initialized. This is explained firther in the next section.

If bit 11 is set to 1, the file system's boot code should have initialized the AppleTalk priority vector. GLoader normally does this as part of the boot process but will not do it if the bit flag is set.

Boot Driver Support

If the boot code jumps to the new GLoader entry point with bit 12 of the A register (the boot driver flag) set to 1, GLoader and GQuit will attempt to load the file */SYSTEM/DRIVERS/BOOT.DRIVER before initializing GS/OS and the start FST. The BOOT.DRIVER file must have a filetype of $BB and an auxtype of $0181. If the file doesn't exist or if the filetype or auxtype is wrong, a fatal error will be reported.

If the original GLoader entry point is used, or if the boot driver flag is 0 on entry to GLoader, the BOOT.DRIVER file will not be loaded.

OS Notification Routine Support

GQuit now calls the Post_OS_event system service routine when switching to ProDOS 8 and after restarting GS/OS.

New Prefix Support

A new prefix has been added to GS/OS. Prefix @ is now set by GQuit before launching a GS/OS application. If the application resides on an AppleShare volume, prefix @ is set to a pathname specified by the AppleShare FST. This will usually be the pathname of the user's folder on the user volume. If the application resides on a non-AppleShare volume, prefix @ is set to the same pathname as prefix 9.

Application Loading Changes

When GQuit calls the System Loader to load a GS/OS application, it now first requests that the application be loaded into non-special memory only. If this attempt fails, it then calls the System Loader again but this time allows special memory to be used. GQuit used to always allow special memory to be used when loading applications, but recent changes to the Memory Manager have increased the possibility of applications being loaded into Super Hires screen memory.

Express Loader Changes

GLoader now attempts to load and execute the */SYSTEM/EXPRESSLOAD file before initializing GS/OS. If the file does not exist, no error is reported. The file remains in memory during OS switches and is not reloaded when switching from ProDOS 8 to GS/OS. GQuit notifies ExpressLoad when OS switches occur.

GS.OS.DEV Support

GLoader and GQuit now attempt to load and startup the */SYSTEM/GS.OS.DEV file before initializing GS/OS. If the file does not exist, a fatal error is reported. This file contains the Device Manager and part of the Device Dispatcher. The file remains in memory during OS switches and is not reloaded when switching from ProDOS 8 to GS/OS.

I/O Redirection Support

In order to support I/O redirection, prefixes 10, 11 and 12 are now defined to be the Standard I/O prefixes as follows -

prefix 10 - StdIn
prefix 11 - StdOut
prefix 12 - StdError

Also, the **QUIT** call's Quit Flag parameter has been modified. Bit 13 of the Quit Flag parameter is now the 'skip_std' flag. The new bit flag is valid for class 0 and class 1 QUIT calls. If the 'skip_std' flag is clear, GQuit will set the Standard I/O prefixes to '.CONSOLE' before launching a GS/OS application. If the 'skip_std' flag is set, the Standard I/O prefixes will remain unchanged from the previous application. When a GS/OS application is launched at boot time or after a P8 application has quit, the Standard I/O prefixes will always be set to '.CONSOLE'.

Splash Screen Changes

The color palette table used by the graphics splash screen code has been modified in order to get rid of the green flash which was occuring just before the Finder desktop appeared and also so that CDEV icons will appear in the correct colors. When the thermometer is drawn, the background is now the same as the Finder background instead of matching the border color.

More Init File Changes

When GLoader loads and executes an init file (from the SYSTEM.SETUP directory), it will now allocate a 4K zero page/stack segment for the init file to use if the file did not request its own zero page/stack segment from the system loader. The segment is deallocated after the init file has executed.

Reserved Bank $01 Space

The address space from $01/BC00 - $01/BFFF is now reserved for use by GS/OS and is therefore not available to GS/OS applications.

.CONSOLE Check

GQuit now checks to make sure that the .CONSOLE device exists before launching a GS/OS application. If it does not exist, a fatal error is reported.

512K System Support

GLoader no longer loads the EXPRESSLOAD file when running on a 512K system in order to free up more memory for application use.

FST Installation Changes

Error handling during FST installation has been changed. It is now a requirement that the initialization routines of all FSTs must handle non-fatal errors themselves and then return carry set and A=0 to indicate that a non-fatal error occurred. GLoader will then unload the FST which encountered the error and continue execution. GLoader will not report the non-fatal error to the user since it is now the FST's responsibility to handle non-fatal errors. This has been done so that FSTs can display specific non-fatal messages during initialization (such as "AppleTalk not turned on") and so that non-fatal errors will not halt the boot process. Fatal errors are handled as before.

ProDOS 8 Compatibility Changes

A "GS/OS compatibility" byte has been added to ProDOS 8. GQuit now checks this byte instead of checking the ascii value of the version number on the ProDOS 8 splash screen. This means that GS/OS is no longer tied to the actual version number of ProDOS 8 or to the location of the version number in the ProDOS 8 code segment.

OS Switching Changes

Major changes have been made to the OS switching mechanism so that when GQuit is switching from ProDOS 8 to GS/OS, it can reload GS/OS from memory instead of from disk.

When quitting from a GS/OS application to a ProDOS 8 application, GQuit now informs the rest of the operating system that a "warm" shutdown should be done instead of a "cold" shutdown. GQuit then attempts to copy the pieces of GS/OS which reside in banks $00 and $01 into non-special memory before launching ProDOS 8. When quitting from a ProDOS 8 application to a GS/OS application, GQuit copies the GS/OS pieces back into banks $00 and $01. GQuit then informs the rest of the operating system that a "warm" startup should be done instead of a "cold" startup. If GQuit is unable to obtain enough memory when switching from GS/OS to ProDOS 8, then it will force a system restart when switching from ProDOS 8 to GS/OS.

Boot File Changes

Because GS/OS is now only loaded from disk during the initial boot, the boot file is no longer saved in memory. As a result, the boot file no longer needs to be divided into temporary and permanent sections and the entry at offset 10 in the jump table is no longer used.

The boot file is described in detail in the "System Startup From Non-ProDOS Devices" section of the "GS/OS GLoader & GQuit External ERS".

More OS Switching Changes

GQuit now makes a copy of the P8 file after loading it from disk. The copy is in non-special memory and is purgable. On subsequent switches to ProDOS 8, the file is loaded from memory instead of from disk unless it has been purged.

"GS/OS Aware" Support

GQuit now checks a GS/OS application's auxtype to detemine if it's "GS/OS aware" before attempting to launch it. If the application is not "GS/OS aware"

and the path to the directory containing the application is more than 64 characters long, then GQuit displays a warning message since the application may not be able to run correctly. If the quitting application was placed on the quit-return stack, then the warning message will allow the user to continue the launch or cancel it and relaunch the application which just quit. If the quitting application was not placed on the quit-return stack, the warning message will allow the user to continue the launch or restart the system. The warning message will only be displayed once for an application; the next time the application is launched the message will not be displayed (unless the application's pathname has changed or the system has been restarted).

Cache Manager Support

GQuit now notifies the cache manager before launching a GS/OS application so that the cache will be purged if the system runs out of memory.

Slot Arbiter Support

GQuit now calls the Slot Arbiter when switching to/from ProDOS 8 in order to save/restore the slot 3 screen holes.

High Sierra File System Translator for GS/OS
External Delta ERS

Version 0.02

By Bryan Atsatt

Revision History

Date	Version	Who	Description
1/31/89	0.01	BPA	Report name error for system service call $01FC54.

About This Document

This document describes changes and enhancements made since the 4.0 system disk release.

Cache_Flsh_Def ($1FC54)

The System Service Calls external ERS v0.11a01, page 4, incorrectly lists the call at $1FC54 as Cache_Lock. The call description on page 17 is correct.

Revision History

Date	Version	Who	Description
3/10/89	0.01	BPA	Describe changes to FST for Extended Attribute Records on directories.
4/6/89	0.02	BPA	Described changes to FST for supporting Apple Extensions on CD-ROM XA discs.

About This Document

This document describes changes and enhancements made since the 4.0 system disk release.

Extended Attribute Records on Directories

The High Sierra and ISO 9660 file systems allow for the presence of Extended Attribute Records (XARs) on both files and directories. The High Sierra FST supports XARs on files but not on directories. When an XAR is encountered on a directory, the FST will return a dir_error ($51).

Apple Extensions to ISO 9660 and CD-ROM XA Discs

The recently published specification for CD-ROM XA includes an incompatability with the Apple Extensions to ISO 9660; the solution, agreed to by engineers at Microsoft and Apple, requires that receiving systems (i.e. the High Sierra FST) handle XA discs as a special case (for details of the format, see the attached document). The High Sierra FST now supports both version 1 and version 2 of the Apple Extensions on CD-ROM XA discs.

ProDOS FST Delta

version 0.02

By
Rob Turner

Revision History:

version 0.01 03/02/89	first version.
version 0.02 03/27/89	added FST specific calls

The ProDOS FST (Pro.FST) has several new features over it's 2.0 version.

(1) Version 2.0 of the ProDOS FST did not write a dirty bitmap to disk until the last file on the dirty volume had been closed. The new version of the FST now writes all dirty blocks to disk when the last file that has been modified is closed. This means that files opened for read access do not affect when the bitmap and directory blocks are updated. This new feature is very important since the GS resource manager leaves the system resource file open until the system is rebooted!

(2) Another new feature of the ProDOS FST is it's ability to add a resource fork to a standard file. This new feature allows files to be retrofitted with a resource fork. To extend a file, the application has to set the storage type to $8005 (Resource + $8000(extend bit)). This new feature brings along with it two new error codes. The error codes are $70 (resource_exist: the file already contains a resource fork) and $71 (res_add_err: The file cannot have a resource fork added, e.g. a subdirectory).

(3) The ProDOS FST will now only display the "Damaged Volume" message once.

(4) ProDOS FST uses the cache manager more often. This means better performance.

(5) The option list is now used by the ProDOS FST. If the option list pointer is setup the ProDOS FST along with all new FST will store the FST ID in the first word following the return count.

03/27/89

Below is a list of FST specific calls that the ProDOS FST supports.

(1) call number $0001: SetTimeStamp

This call allows the user to set the time stamp option. This call affects the performance of the ProDOS FST. By default the ProDOS FST time stamps all files and all subdirectories to the roc level. This means that a change to a file in a subdirectory is reflected in the modification date of each parent subdirectory all the way to the root level of the volume. Consider the following example, which copies a file from some source to the destination path ":Dest:Subdir1:Subdir2:Subdir3:File".

```
create  ":Dest:Subdir1:Subdir2:Subdir3:File". (Time stamp: File,Subdir3,Subdir2,Subdir1)
copy data here loop
close   ":Dest:Subdir1:Subdir2:Subdir3:File". (Time stamp: File,Subdir3,Subdir2,Subdir1)
SetInfo ":Dest:Subdir1:Subdir2:Subdir3:File". (Time stamp: File,Subdir3,Subdir2,Subdir1)
```

Call structure:

```
        PCount       $0003
        FST_ID       $0001 (ProDOS FST)
        Call_Number  $0001 (SetTimeStamp)
        Time_Option  $xxxx (input: Listed in table below)
```

Time_Options:

$0000 =	Time stamp files only	(Fastest)
$0001 =	Time stamp file + Parent Directory	(Slower)

$0002	=	Time stamp file + all directories to the root	(Slowest)

(2) call number $8001: GetTimeStamp

This call allows the user to get the time stamp option.

Call structure:

PCount	$0003
FST_ID	$0001 (ProDOS FST)
Call_Number	$8001 (GetTimeStamp)
Time_Option	$xxxx (reseult: see table below)

Time_Options:

$0000	=	Time stamp files only	(Fastest)
$0001	=	Time stamp file + Parent Directory	(Slower)
$0002	=	Time stamp file + all directories to the root	(Slowest)

(3) call number $0002: SetCharCase

This call allows the user to change the way that the ProDOS FST saves and returns file names. the caller wishes the ProDOS FST can save the file name with upper and lower case. If a subdirectory name is saved with lower case letters it may not be accessible under older version ProDOS 8, to fix this problem use the version of ProDOS 8 on system disk 5.0 or greater, or rename the subdirectory to all upper case.

Call structure:

PCount	$0003
FST_ID	$0001 (ProDOS FST)
Call_Number	$0002 (SetCharCase)
Case_Option	$xxxx (input: see table below)

Case_Option:

$0000	=	Turn OFF upper/lower case
$0001	=	Turn ON upper/lower case

(3) call number $8002: GetCharCase

This call allows the user to get the current case settings.

Call structure:

PCount	$0003
FST_ID	$0001 (ProDOS FST)
Call_Number	$8002 (GetCharCase)
Case_Result	$xxxx (result: see table below)

Case_Result:

$0000	=	lower case is disabled.
$0001	=	lower case is enabled.

SCC Manager ERS

EXTERNAL

Version 0.06

By Tim Harrington

Change History

Date	Version	Who	Description Of Changes
14-Sept-88	0.00	TGH	Initial Document
25-Oct-88	0.01	TGH	Added information to example. Added information about AppleTalk drivers.
3-Jan-89	0.02	TGH	Changed name from AppleTalk Supervisor Driver to SCC Manager. Added two new channel allocation calls.
14-March-89	0.03	TGH	Added channel number to AppleTalk client call.
27-March-89	0.04	TGH	Changed references to *Universe*.
13-April-89	0.05	TGH	Added two equates (slot_num and unit_num) to the sample code.
25-April-89	0.06	TGH	Fixed bug in sample code.

About This Document

This document covers the design goals and calling mechanism for the SCC supervisor driver. It assumes the reader already has knowledge of the following:

- GS/OS device management.

Background

Previous to the *Universe* system disk, the AppleTalk protocols were loaded by a system init program. It was only after this program would load and initialize the AppleTalk protocols that another system init or program could use AppleTalk. With *Universe* we needed a means to use AppleTalk before the system init programs get a chance to run. Therefore the AppleTalk load procedure had to be changed.

Our new requirements were that the AppleTalk protocols must be loaded and initialized before any standard GS/OS driver was initialized. This was accomplished through the use of a GS/OS supervisor driver which this document describes.

Because GS/OS is a modular based operating system, a method of arbitrating system resources between different modules is necessary. This driver also provides the arbitration for the two channels of the SCC.

Overview

The SCC supervisor driver is a loaded driver which provides a supervisory service to the AppleTalk protocols, GS/OS AppleTalk drivers, and any other GS/OS driver that requires the use of the SCC. The driver provides the following services:

- **Loading of the AppleTalk protocol files.**
  The AppleTalk supervisor driver both loads and initializes all the available AppleTalk protocols found in the */SYSTEM/DRIVERS directory.

- **GS/OS AppleTalk driver unit and slot number assignment.**
  When an AppleTalk GS/OS driver starts up, it must call the supervisor driver to determine which slot AppleTalk resides in and request a unique unit number for that slot.

- **SCC channel arbitration.**
  This allows services such as AppleTalk, MIDI, Modem, and Printer drivers to effectively share the SCC.

Device Driver Calls

SUPERVISOR STARTUP $0000

This call initializes the supervisor driver. This call will be issued by GS/OS on supervisor startup time. An application must NOT issue this call.

When this call is issued, the supervisor driver will find the AppleTalk slot. If there is no AppleTalk slot, it returns with no error.

If an AppleTalk slot is found, the driver would then begin to load the protocols files in the DRIVERS subdirectory. It loads and initializes the protocol files in alphabetical order. The protocol files *MUST* be named such that they will be loaded in the proper order.

SUPERVISOR SHUTDOWN $0001

This call resets the SCC to its hardware reset state.

APPLETALK CLIENT $0002

This call returns to the calling driver the AppleTalk slot number, channel number and a unique unit number which that driver should use in its DIB.

This call uses the following parameter block:

Offset	Field
0–1	Slot number
2–3	Channel number
4–5	Unit number

Slot number - The supervisor driver will store the AppleTalk slot number here.
Channel number - The supervisor driver will store the AppleTalk channel number here.
Unit number - The supervisor driver will store a unique AppleTalk unit number here.

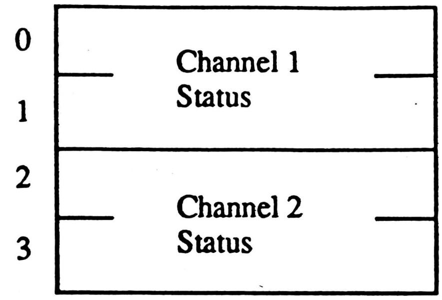
The calling driver must have put a pointer to the parameter block on the direct page at $78 as is defined in the GS/OS supervisor driver documentation.

GS/OS drivers are expected to ask for a new unit number every time they get started up. This means that when switching from P8 back to GS/OS the driver should re-issue the AppleTalk Client call in order to receive a new unique unit number..

GET CHANNEL STATUS $0003

This call returns the current status of the SCC channels.

This call uses the following parameter block:



Channel 1 Status - $0000 = Channel available.
$0001 = Channel being used.
$0003 = Channel being shared

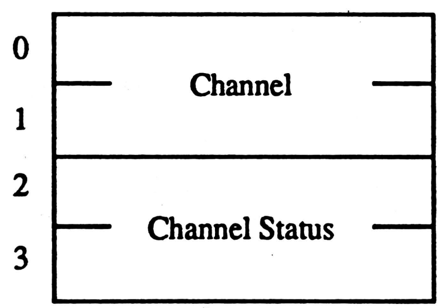
Channel 2 Status - $0000 = Channel available.
$0001 = Channel being used.
$0003 = Channel being shared

The calling driver must have put a pointer to the parameter block on the direct page at $78 as is defined in the GS/OS supervisor driver documentation.

SET CHANNEL STATUS $0004

This call sets the status of the specified SCC channels.

This call uses the following parameter block:



Channel - $0000 = Channel 0.
$0001 = Channel 1.

Channel Status - $0000 = Channel available.
$0001 = Channel being used.
$0003 = Channel can be shared.

The calling driver must have put a pointer to the parameter block on the direct page at $78 as is defined in the GS/OS supervisor driver documentation.

The channel status is maintained only between the GS/OS startup and shutdown calls. After switching back from P8 to GS/OS, the channel status for both channels, with the exception of the AppleTalk channel, are re-set to 0 (channel available.)

Possible errors:

drvr_bad_parm	$22	Bad call parameter
drvr_no_resrc	$26	Resource not available

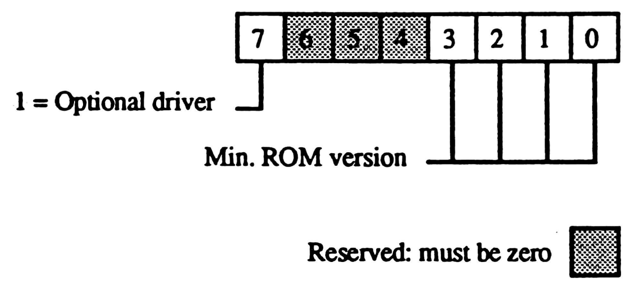
AppleTalk Drivers

The SCC supervisor driver loads and initializes all AppleTalk drivers found in the DRIVER directory on the boot disk. An AppleTalk driver is identified by a file type ($BB - driver file) and the high order order byte of the driver aux type ($02 - AppleTalk driver). The low order byte of the aux type contains information about under what startup conditions the driver should be loaded. The aux type is defined as follows:

High byte:

The high byte must be $02.

Low byte:



An AppleTalk driver will only be loaded if the Min. ROM Version is greater than or equal to the version of the ROM currently installed.

All AppleTalk drivers are loaded when the Apple IIgs is being booted on a local disk drive. If the computer is being booted over the network, only the drivers with the Optional Driver bit set will be loaded.

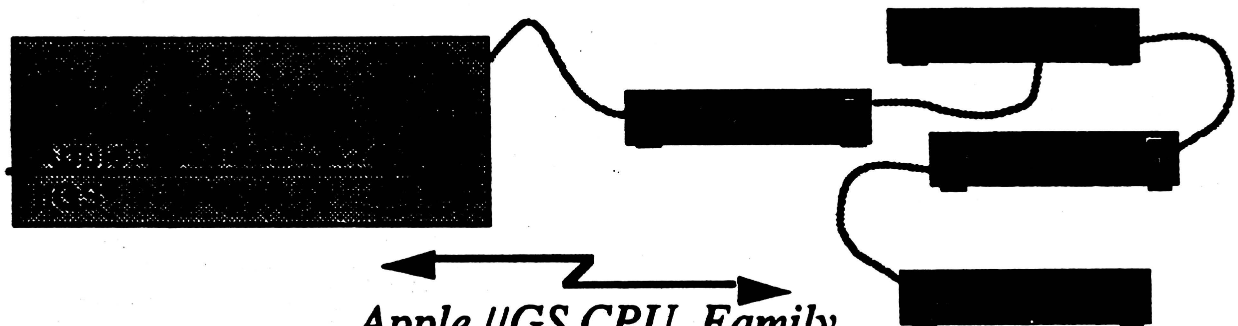
The supervisor driver will load the AppleTalk driver via an InitialLoad call. The supervisor driver will then initialize the AppleTalk driver by jumping to the first byte of the driver. The X register will contain the Apple IIgs ROM version and the Y register will contain the AppleTalk slot number. On exit from the initialization routine, the AppleTalk driver should set the accumulator to zero and the carry clear.

Examples

The following code example allows a standard GS/OS driver to find the AppleTalk slot number and acquire a unique unit number for the slot.

```
slot_num        equ  $2E          ;Offset into DIB to Slot #
unit_num        equ  $30          ;Offset into DIB to Unit #

sup_drvr_disp   equ  $01FCA4      ;GSOS system service vector.
```

```
drvr_dib_ptr       equ   $20                ;GSOS driver zero page.
sup_parm_ptr       equ   $78                ;GSOS driver zero page.

              phk
              plb

              rep    #$30
              longa  on
              longi  on

              lda    #$0000                 ;No driver number (None).
              tax                           ;Call number 0 (GetSupNum).
              ldy    #$0001                 ;Supervisory ID #1 (SCC).
              jsl    sup_drvr_disp
              bcs    Error

              stx    sup_num                ;Store AppleTalk supervisor number
                                            ;for later use.

              lda    #ParmList              ;Store parameter block pointer
              sta    <sup_parm_ptr          ;on zero page for supervisor's
              lda    #^ParmList             ;use.
              sta    <sup_parm_ptr+2
              lda    sup_num                ;Supervisor driver number.
              ldx    #$0002                 ;Call number 2 (AppleTalkClient)
              ldy    #$0001                 ;Supervisor ID #1 (SCC)
              jsl    sup_drvr_disp
              bcs    Error

              ldy    #slot_num              ;Slot number offset
              lda    Slot
              sta    [drvr_dib_ptr],y       ;Store slot # in our DIB
              ldy    #unit_num              ;Unit number offset
              lda    Unit
              sta    [drvr_dib_ptr],y       ;Store unit # in our DIB

              clc
Error         rts                           ;C clear = no error
                                            ;C set = no AppleTalk

sup_num       ds.w   1                      ;Supervisor driver number goes here

ParmList                                    ;Parameter list for AppleTalkClient.
Slot          dc.w   0
Channel       dc.w   0
Unit          dc.w   0
```



Apple //GS CPU Family SCSI MANAGER

EXTERNAL ERS

Written by: Gus Andrade
and Matt Gulick

Version 0.19

Preliminary

Revision History:

Date	Version	Description of Revision
4-4-88	0.01 Preliminary	Document provided to Mike A for inclusion in Alpha documentation.
4-5-88	0.02	1. SCSI Manager Device and SCSI I/O calls defined 2. Block Sparing on devices will be done at driver. 3. SCSI Device partitioning will be done at driver. #2 & #3 are device specific and should be done by the individual drivers.
4-11-88	0.03	1. Get_Devices now tells SCSI Manager to check for SCSI device types as defined by the SCSI Standard instead of GS/OS SCSI device type. 2. The list returned to the caller on Get_Devices call provides no longer contains the number of bytes in the buffer.
4-13-88	0.04	1. The buffer size of Get_Devices has grown from $055C to $0700 bytes in size. 2. A new section has been added giving the bit breakdown for error codes.
4-26-88	0.05	1. Added Intercommunication vector to Get_devices call to permit the SCSI manager notifying a driver that a something has happened to the device environment controlled by a driver. 2. Fixed errors between DEVICE I/O calls and SCSI I/O regarding the parameter list discrepancies.
5-20-88	0.06	1. Added a reserved Long word field to the SCSI DEVICE and SCSI I/O calls. Also, added a commands flag word to SCSI DEVICE and SCSI I/O calls.
6-2-88	0.07	1. Moved RESERVED fields in SCSI DEVICE & SCSI I/O calls to the end of data structures. 2. Added Status Ptr to the command list data structure and also added two reserved long fields for future growth.
6-2-88	0.08	1. Fixed error in Get_Devices call where it states that $0700 bytes are required has been changed to say that $0702 bytes are required.
7-25-88	0.09	1. Renamed Get_Devices call to Request_Devices. Has redefined the device type field. 2. Size of Request_Devices data buffer changed from $0702 to $0704 bytes in size. 3. Created calls: Claim_Devices 4. The Status buffer pointer points to a one word field.

		5. Added a long word entry for Message buffer pointer to the SCSI Device and SCSI I/O calls.
7-26-88	0.10	1. Added a Time-out field to the parameter list for Device call and I/O call to prevent the SCSI Manager from hanging indefinitely.
8-1-88	0.11	1. Updated the Device types list to abide by the SCSI 2 version of the standard.
9-7-88	0.12	1. Modified the parameter list of the DEVICE & I/O calls in major ways 2. Modified the SLOT field in the REQUEST & CLAIM DEVICES calls to include the logical unit number. 3. Modified the structure of the send and receive buffers list to incorporate a looping capability in the specification of buffers pointers. This change has been made to both the SCSI Device & I/O call. 4. The DEVICE call is to be issued when the number to be transferred from the target can be less than the requested byte count.
9-29-88	0.13	1. Major revision of the SCSI Manager ERS. a. Clean-up in the introductory section b. Removed SCSI Device call c. Expanded explanation of the fields in all calls. d. Major revision of parameter list for I/O call.
10-2-88	0.14	1. Changed the Stat/Msg buffer pointer to AutoSense buffer pointer. 2. Removed the AutoSense enable bit in the call attributes word. Attributes word now reserved.
10-27-88	0.15	1. Added a new pointer in the SCSI I/O parameter list to support messages from the caller. 2. Added a bit in the attributes word of the SCSI I/O call to notify the SCSI Manger that the calling wants to allow the target to disconnect during a call.
03-21-89	0.16	1. Added error code changes to all the calls. 2. Modified the definition of the ID field to conform with the current definition of slot & unit numbers for DIBs. 3. Generalized cleanup.
04-24-1989	0.17	Removed reference to linked commands.
05-12-1989	0.18	Fixed Picture of SCSI Data Model.

05-19-1989	0.19	Clarified Explanation of Data supplied in the command Descriptor Block. Also updated explanation of the notification vector.

Issues Page:

Date	Issue Description
4-6-88	1. Should we allow partitions to be added to a partitioned device without forcing formatting of the entire device? (Issue for drivers)
	2. How about installing SCSI devices as the come on-line. In other words issuing Post_Driver_Install calls for SCSI devices which are powered on after the system is running.
4-13-88	1. A mechanism to notify the SCSI Manager that the calls issued are I/O (read/write) calls from or to a block device. This is required if the SCSI Manager will provide Caching support for blocks on the device.(Answer: Driver does Cache support)

Introduction

SCSI (Small Computers Systems Interface) is a communications protocol standard which defines the interaction between peripheral devices and computer systems over a common data bus. It specifies rules for the arbitration, selection and data transfer between an initiator and a target device. The standard also defines error reporting mechanisms, hardware and cabling requirements between initiator and target devices

Due to the general nature of the SCSI standard it permits a wide range of devices to be connected on the SCSI data bus. SCSI supports block storage devices such as hard disks CD roms and tape drives. It also supports character devices like printers and scanners.

The remainder of this document deals with the SCSI Manager implementation for the Apple // family of products running under GS/OS.

Current Apple // SCSI implementation.

The current SCSI implementation was designed two years ago for the ProDOS 8 (ProDOS 16 is the same as ProDOS 8 for this discussion since it makes ProDOS 8 calls) operating system. ProDOS 8 is an efficient operating system written for the IIe and IIc computers. ProDOS 8 and the ProDOS calls to peripheral cards were sufficient for the early 1980's but not for the late 1980's and early 1990's. GS/OS is the operating system for the IIgs and for the Apple II line in the 1990's. GS/OS is a high performance operating system. As such it has more efficient and better ways to deal with peripheral cards than the ProDOS interface. The SCSI card / SCSI card firmware was designed for the old operating system. The code is not sufficient for the new high performance operating system GS/OS.

Limitations of current Apple IIGS SCSI implementation.

- ProDOS file limitation of 16 megabytes maximum.
- ProDOS volume size limitation of 32 megabytes maximum.
- 32 megabyte volume size limitation forces very large hard disks to be into multiple 32 megabyte volumes.
- No consistent communication interface, ie; ProDOS and SmartPort entry points.
- Double buffering of data through bank $00.
- Lack of interrupt support will not permit issuing Asynchronous SCSI calls.
- No consistent mechanism for issuing SCSI device specific calls.
- Firmware forced to run at slow system speed because it resides in slow side of the system.

Enhancements in GS/OS SCSI Manager.

- One consistent communication interface between drivers and the hardware.
- Isolation of hardware from the operating system (access through drivers only) The operating system does not access hardware directly.
- No double buffering of the data
- Support for current Apple II SCSI card. (Without IRQ or DMA support).
- Interrupt support in systems with built NCR 5380.
- DMA support to 1 megabyte/sec or greater.
- SCSI Manager able to run at full system speed except when accessing I/O soft switches.

GS/OS based SCSI Supervisor Driver. (SCSI Manager)

The SCSI manager will control all accesses through the NCR 5380 SCSI controller chip. It will be responsible for arbitration of hardware resources and manage access to partitions on specific SCSI devices. The manager will run on the IIGS with a minimum of 512 K of system memory. Application programs will call device drivers which in turn call the SCSI Manager. This way the SCSI Manager level can change in the future without affecting applications or device drivers and the drivers will be hardware independent.

The current SCSI card does not support interrupts or DMA. Future versions of the SCSI manager will support interrupts and DMA transfers in systems which can allow them.

SCSI Manager Architecture

The SCSI manager controls all communications on the SCSI bus and manages the arbitration between drivers and SCSI peripherals. The following diagram shows the position for the SCSI Manager relative to drivers in the GS/OS scheme of things. It also gives a breakdown of possible SCSI devices.

The SCSI Manager knows about SCSI peripheral cards, physical SCSI devices and logical SCSI devices only. It does not know anything about partitions. In order to access a partition on a physical SCSI block device or a Logical SCSI block device, the driver must set the block number such that it points to a block within a specific partition. (translate logical block number to physical block numbers).

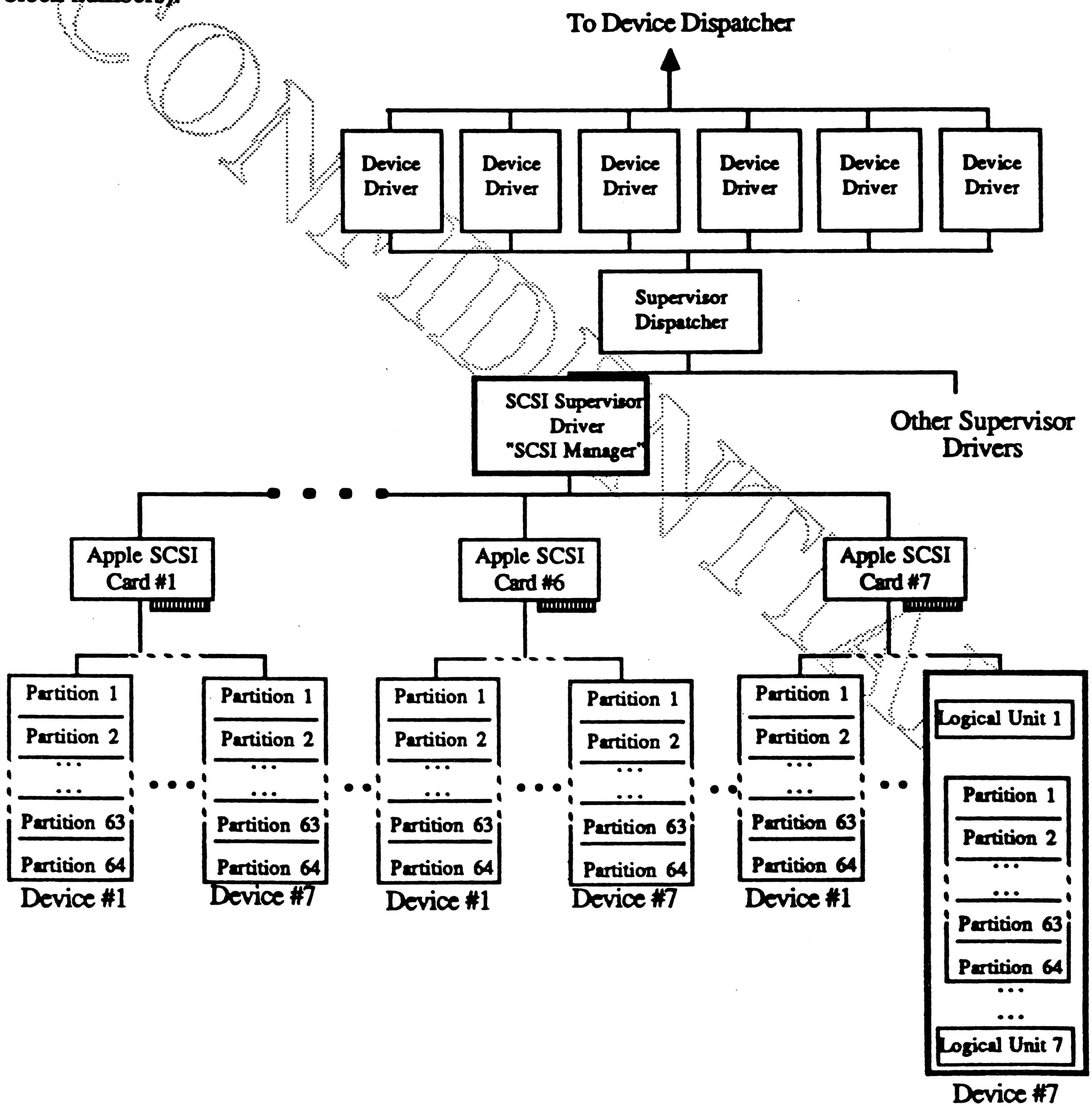


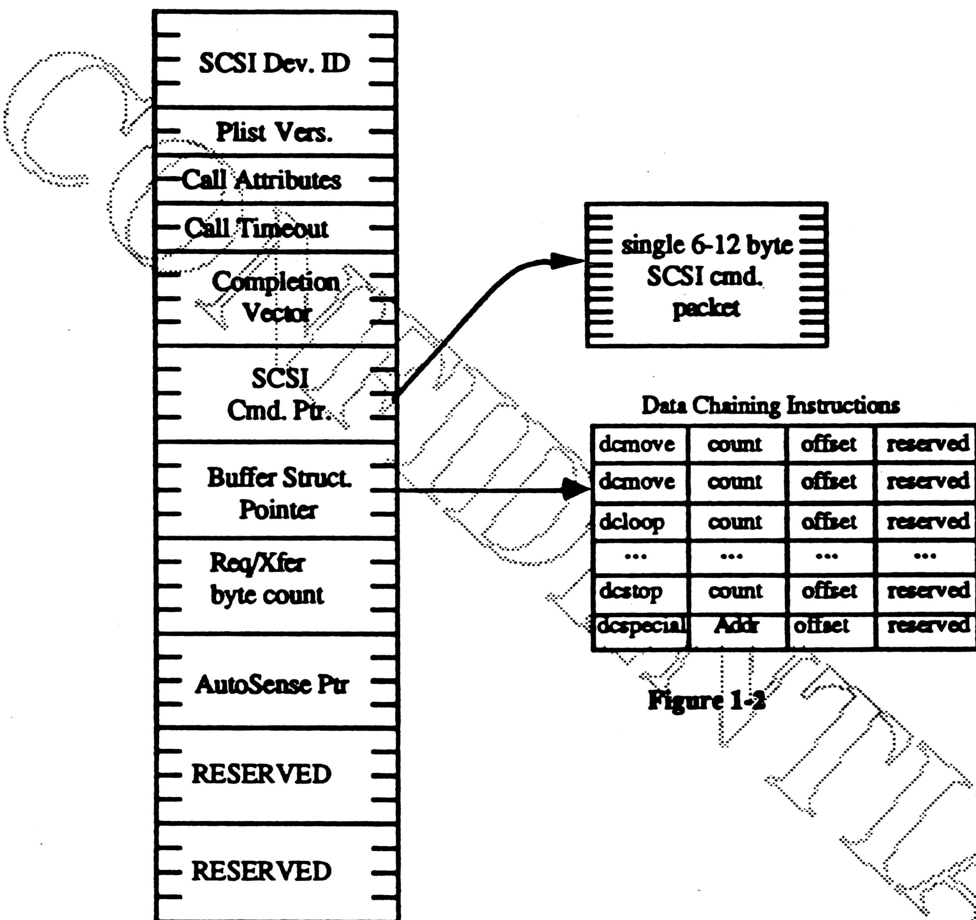
Figure 1-1

The SCSI manager supports the current Apple // SCSI card. There can be seven cards plugged into a system and each card can support seven devices. Although the SCSI Manager does not know about partitions, room has been reserved in the device id field to allow device drivers to access a maximum of 64 partitions. Therefore even though there are only seven possible SCSI target devices on the SCSI bus, there can be many more available through multiple cards and partitions. Refer to Figure 1-1 for a pictorial representation of possible devices the SCSI manager supports. It will send commands and data to SCSI devices in pseudo DMA mode or hardware DMA mode depending on the type of card. The following list contains the major areas the SCSI Manager will control.

- Hardware resources management.
    1. Support startup and shutdown as defined by the Supervisor Dispatcher.
    2. Support call interface between SCSI drivers and the SCSI manager.
    3. Built-in NCR 5380 support.
    4. Support for current SCSI card implementation.
    5. Low level phase support to access the SCSI bus, select a target, transfer data to and from the device.
        a. Bus free Phase
        b. Arbitration Phase
        c. Selection Phase
        d. Reselection Phase
        e. Command Phase
        f. Data Phase.
        g. Status Phase
        h. Message Phase
        I. SCSI bus reset

SCSI Data Model

Besides the required calls for Supervisory drivers there is one call to access SCSI target devices. The SCSI I/O call handles all communication between initiators and targets. Device drivers make the I/O call with a set parameter list and the SCSI Manager will manage the communication between the system and the target. The parameter list has the following format:



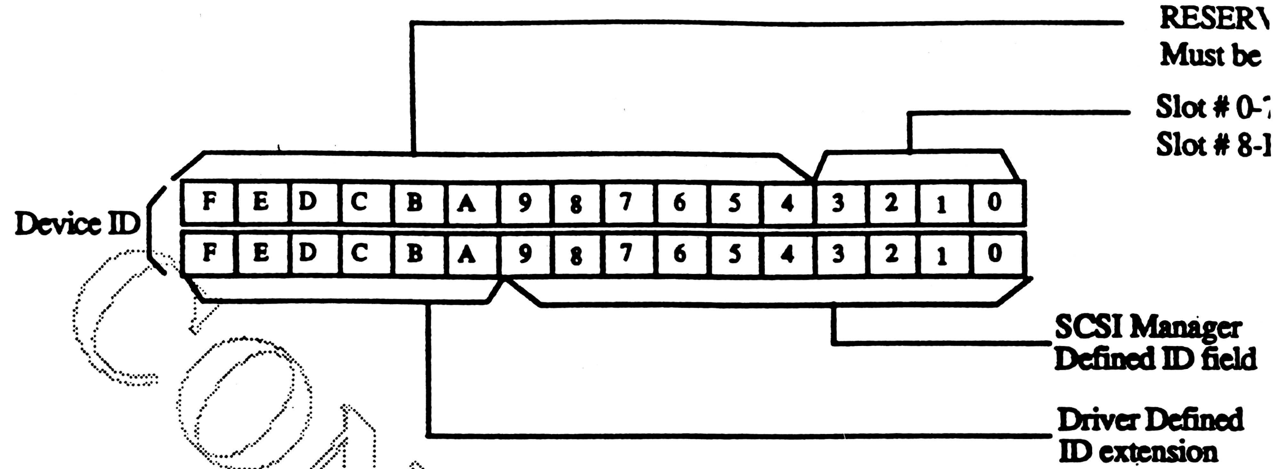
Data Chaining Instructions

dcmove	count	offset	reserved
dcmove	count	offset	reserved
dcloop	count	offset	reserved
•••	•••	•••	•••
dcstop	count	offset	reserved
dcspecial	Addr	offset	reserved

Figure 1-2

SCSI Device ID:

This long word wide field contains the ID given devices by the SCSI Manager. The ID is defined by the SCSI Manager for physical SCSI devices and logical units within SCSI devices. The Driver Defined ID Extension is used by block device drivers to differentiate partitions on block devices. The Driver Defined ID extension is set to zero and should not be modified by the driver unless the physical or logical device contains partitions on it. If the device contains partitions, the Driver defined ID extension must be unique for each partition on that device. When a call is made to the SCSI Manager the driver must set bits A-F to zero.



Call Attributes:

This field is a bit encoded word describing how the I/O call is to be handled. Refer to the I/O call for a description of the bits.

Version:

This field contains a version word defining the type of parameter block. This version field must be set to zero.

Call Time-out:

This field contains the number of 1/4 second tics which must expire before the I/O call issued is aborted. If the call does not complete within this specified time and error will be returned to the driver.

Completion Vector:

When a call completes, the SCSI Manager will call the driver through this vector to notify completion of a call.

SCSI Cmd. Ptr:

This field points to a single or multiple SCSI command packets. Each packet can be six, ten or twelve bytes in size. The Data contained in the command packet is structured exactly as described by the SCSI Spec with all fields containing data in the format expected by the target device.

Buffer Structure pointer:

This long pointer contains the address of a list of data chaining instructions. Refer to the Data Chaining section for a detailed description.

DATA Chaining:

Data Chaining is a mechanism which allows the specification of multiple source or destination buffers without the need to have one entry for each buffer address. There is a maximum of 32 data chaining instructions for any call issued to the SCSI Manager. The format of data chaining instructions is as follows:

Opcode	Count	Offset	Save Pointer
Long Word	Long Word	Long Word	Long Word

The **Opcode field** defines the type of instruction. There are three data chaining instructions defined:

1. DCMOVE := Any value other than zero or negative one.
   This instruction is any value which is not a DCLOOP instruction or a DCSTOP instruction. The long word value in the opcode field will be treated as a buffer address. Bits 24-31 of the buffer address will be ignored and should be set to zero.

2. DCLOOP := -1. ($FFFFFFFFFFFFFFFF)
   This instruction passes control to another data chaining instruction.

3. DCSTOP := 0. ($0000000000000000)
   This instruction tells the SCSI Manager that there are no more data chaining instructions to execute.

4. DCSPECIAL := If the Count field in the DCSTOP instruction is nonzero this field contains an address through which the Manager passes control to notify the driver that the manager has reached this instruction. The Offset field of the DCSPECIAL instruction can be set to any value the caller chooses to identify which DCSPECIAL instruction is currently running.
   WARNING: The driver must return back to the Manager with the system environment set exactly as it was when the Manager passed control to the driver.

The **Count field** defines the number of bytes to transfer if the data chaining instruction is a DCMOVE instruction. If the instruction is a DCLOOP then it defines the number of times this loop instruction will be executed. If the opcode is a DCSTOP then the Count field must be set to zero. The instruction will be a DCSpecial if the Opcode field is a zero and the Count field is nonzero. In this case the count field becomes an address through which control is passed to the driver when the manager encounters a DCSPECIAL instruction.

The **Offset field** contains the value to add to the buffer address if the instruction is a DCMOVE instruction. If the opcode is a DCLOOP, the Offset defines the relative offset from the current data chaining instruction number to branch to. If the opcode is a DCSTOP the Offset must be set to zero.

The **Save pointer** field is reserved for use by the SCSI Manager. It should be set to zero by the caller. The field will contain the current buffer address after a Save pointers message is received from a target device.

Table summary of data chaining instructions:

Opcode	Count	Offset	Save Pointer
Long Word	Long Word	Long Word	Long Word
DCMOVE (Buff Addr)	Buffer Size	Value added to Buffer Addr	RESERVED* Set to zero
DCLOOP (Loop Opcode)	Loop Count	Relative Offset to another instruction.	RESERVED* Set to zero
DCSTOP (Stop opcode)	Zero*	Zero*	Zero*
DCSPECIAL (Stop opcode)	Address	Zero*	Zero*

* These fields reserved for use by the SCSI Manager. Must be set to zero.

The following example reads 16 groups of $800 bytes each and stores them in buffers which have starting addresses $3000 bytes apart from each other, starting at buffer address $2000.

Inst. No.	Opcode	Byte Count	Offset	Save Ptr.
1	$00002000	$00000800	$00003000	$00000000
2	$FFFFFFFF DCLOOP	$00000010	-1	$00000000
3	$00000000 DCSTOP	$00000000	$00000000	$00000000

SCSI Manager Calls

The SCSI Manager will support calls as defined by the Supervisor dispatcher ERS. Refer to the Supervisor Drivers section in the GS/OS Device Drivers ERS for a detailed description of Supervisor Drivers and calling procedure.

The calls defined for GS/OS are defined as follows:

- Supervisor Driver Startup (SCSI Manager Startup call made by Supervisor Dispatcher)
- Supervisors Driver Shutdown. (SCSI Manager Shutdown call made by Supervisor Dispatcher)
- Supervisor Driver Specific Calls. (SCSI Manager calls made by device driver)
    1. Get number of devices.
    2. Claim number of devices.
    3. SCSI I/O call. (generic call for reading or writing to SCSI devices)

When a call is made by a driver, control is passed to the SCSI Manager through a vector defined by GS/OS. When control is returned to the driver at the end of the call, the Accumulator will contain $0000 or an error code which the driver must interpret and translate into GS/OS valid error codes. If an error is returned the carry bit will be set otherwise it must be cleared.

In the case of Startup and Shutdown, the error code in the accumulator is not required, only the carry bit set or cleared will be checked by the Supervisor dispatcher.

SCSI Manager Startup   Call $0000

This call will initialize the SCSI Manager. It allocates memory from the Memory Manager for internal use, checks to see how many devices are on-line and builds its devices list. This call is made by the Supervisor Dispatcher to each supervisor driver loaded. This call can NOT be made by Device drivers. The Parameter list for this call follows:

Call Input Parameters:	A Reg:	SCSI Manager ID	≠ $0000
	X Reg:	SCSI Manager Startup Call #	= $0000
	DirectPage:	GS/OS Direct Page	
Call Output Parameters:	A Reg:	Error Code	

Supervisor Driver Number: Word parameter for the Supervisor Driver number to be started.

SCSI Manager Call #: This word parameter specifies which type of call is to be issued to the supervisory driver.

GS/OS Direct Page: Refer to the GS/OS memory map for a detailed definition of the contents of this page. Locations $78-$7B within the page point to the SCSI Manager SIB pointer.

Error code:

$FE07 = Unable to allocate SCSI Manager internal resources
$FE0B = General SCSI Manager Startup error (Manager gets purged)
$FE0E = SCSI Manager IRQ handler install error
$FE12 = Unable to find DIB for boot slot.

SCSI Manager Shutdown Call $0001

This call will be made to release all resources used by the SCSI Manager. This call will be made by the Supervisor Dispatcher after all SCSI drivers have been shutdown.

Call Input Parameters:	A Reg:	Supervisor Driver Number	≠ $0000
	X Reg:	Supervisor Call Number	= $0001
	DirectPage:	GS/OS Direct Page	
Call Output Parameters:	A Reg:	Error Code	

Supervisor Call Number: This word parameter specifies which type of call is to be issued to the supervisory driver.

Supervisor Driver Number: This word parameter specifies which supervisory driver is to be shutdown.

GS/OS Direct Page: Refer to the GS/OS memory map for a detailed definition of the contents of this page. Locations $78-$7B within the page point to the SCSI Manager SIB pointer.

Error code:
$FE09 = Unable to shutdown SCSI manager

Request SCSI Devices   Call $0002

This call returns the number of devices the SCSI Manager found on-line during the SCSI Manager startup for a specific SCSI Peripheral device type. The SCSI peripheral device type is defined below under SCSI Device Type.

The caller must provide a buffer of $0704 bytes in size. This buffer size assumes a worst case device allocation of eight APPLE SCSI peripheral cards with seven physical SCSI devices on each card with eight logical devices for each physical device.

The information returned to the caller will be made up of one word defining a scratch zero page which the driver can use during the call, one word defining the number of devices found which match the requested device type, one word defining the logical unit number of the target and the slot number, and a one word of ID defined by the SCSI Manager. The ID numbers defined by the SCSI Manager will range between $0001 and $01C0. The slot number is to be inserted in the DIB. The LU# is to be inserted in the command packet by the driver.

Call Input Parameters:	A Reg:	ID for SCSI Manager	≠ $0000
	X Reg:	Get SCSI Devices Call	= $0002
	DirectPage:	GS/OS Direct Page	

GS/OS Direct Page Contents:

Offset	Parameter type	Size
$74	SCSI Manager SIB ptr.	Long
$78	Parameter list pointer	Long

Parameter List:

Offset	Parameter type	Size	Comments
$0000	SCSI Device type	Word	SCSI standard peripheral device type
$0002	InterComm vector	Long	Vector used for SCSI man to notify drivers of environment changes.
$0006	Destination Buffer ptr.	Long	Buffer must be $0704 bytes long.

SCSI Device Type:

The SCSI device type is defined by the SCSI standard. It is a one byte code defining SCSI devices. Bits 0-7 define the device type as specified in the SCSI standard

$00	Direct Access device (Magnetic disk)
$01	Sequential access device (magnetic tape)
$02	Printer device
$03	Processor device
$04	Write-once read-multiple device (e.g., some optical disks)
$05	Read-only direct-access device (e.g., some optical disks)
$06	Scanner Device (Defined in SCSI 2 Standard working draft)
$07	Optical Memory Devices
$08	Changer devices (e.g. Jukeboxes)
$09	Communications Devices
$0A-$0F	Reserved
$10	Apple Tape Drive (3M MCD/40)
$11-$1D	Reserved
$1E	Target Device
$1F	Unknown Device type

1. Check for LU 0 by issuing an Inquiry call to LU 0.

2. Check for LU 7 by issuing an Inquiry call to LU 7.
3. Compare the data from step 1 and 2
4. If the Inquiry data matches, keep LU 0, ignore LU 7 and do not scan for any other logical units under this SCSI Device.
5. If the data does not match then scan LU 1 through LU 6 comparing the Inquiry data looking for a match. Any time the data from two Inquiry calls matches keep the lower LU and do not scan for any other LUs in that SCSI Device ID.

If any bit positions are set to one, only Logical units with corresponding bits set to one will be checked.

InterComm Vector:

This long word field contains the address of the routine the SCSI Manager will pass control to. The structure of the data passed by the manager and what that data means has not yet been defined. Until further notice this field is considered reserved, with no action being taken when non-zero.

Destination Buffer Pointer:

This long word field contains the pointer to a $704 byte buffer which will contain the list of devices requested.

Call Output Parameters:

Registers: A Reg Error Code

Destination Buffer:

Offset	Parameter type	Size	Comments
$0000	Zero page address	Word	Start of zero page usable by scsi drivers
$0002	Total Device Count	Word	# devices matching SCSI device type requested
$0004	Device ID	Long	Bits 0-3 of low word contain Slot # 0-F device belongs to. Slots 0-7 are internal slots Slots 8-F are external slots High word contains the unit number assigned to the device by the Manager. Note the Device ID matches the slot and unit number definition for DIBs.
...	....	...	

Entry with offset $0004 will be repeated the number of times defined by field $0002 (Total Device count).

Zero Page Address:

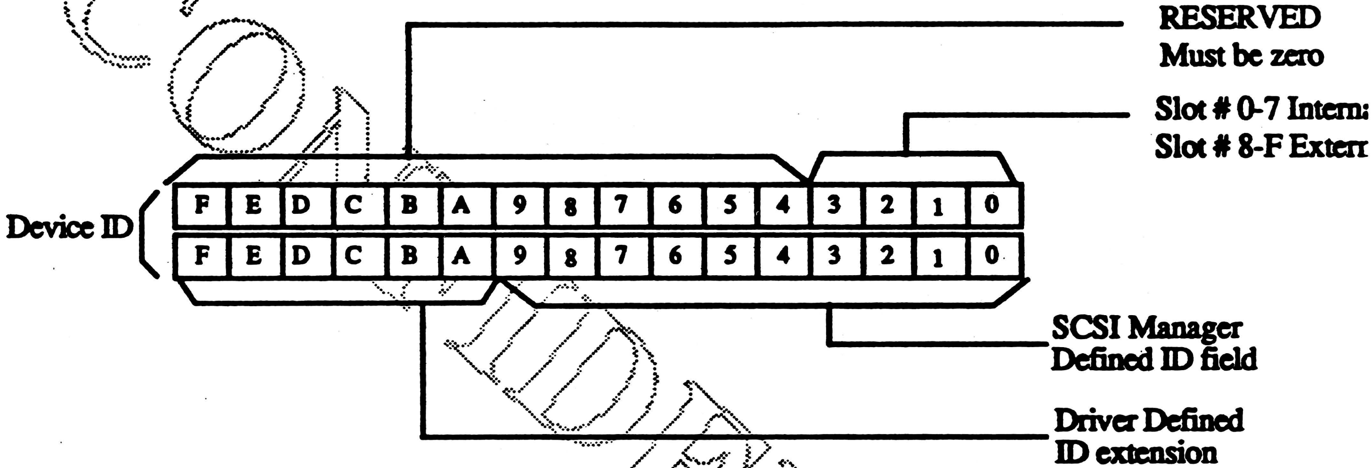
This field contains the address of a scratch zero page which can be used by device drivers calling the SCSI Manager. This zero page is active only during the call and drivers should no count on data remaining valid between calls.

Total Device Count:

This word field contains the number of target devices returned to the caller which match the specified device type.

DEVICE ID:

This field is assigned by the SCSI Manager to identify devices it finds on the SCSI bus. The driver defined ID extension is provided for support of SCSI block devices with multiple partitions. It is set to zero by the SCSI Manager and can be changed by SCSI block device drivers which support partitions to make each partition on a specific SCSI device unique. The SCSI Device ID must be unique. Multiple devices with the same SCSI Device ID are not allowed. The first ID extension must be zero.



Errors:

$FE01 = Invalid GS/OS SCSI device Type

Claim SCSI Devices Call $0003

This call tells the SCSI Manager which devices in the list returned by the request call have been claimed by a driver. The parameter list for this call is identical to the parameter list for the Request Devices call. The buffer pointer will point to a buffer which contains the identical list returned from the Request Devices call with the exception of the high order bit of the slot # field within the Device ID. If the bit is set to a one it marks that device as claimed by the driver. If the bit is zero then that device can be claimed by another driver. Devices handed to drivers by the Request devices call must be claimed by the driver if the driver wants to access the device. If the driver does not claim the device and accesses it there may be a conflict with multiple drivers trying to access the same device and very unpredictable results may occur. The only way to guarantee that a driver has sole access to a device is to claim it.

Call Input Parameters:			
	A Reg:	ID for SCSI Manager	≠ $0000
	X Reg:	Get SCSI Devices Call	= $0002
	DirectPage:	GS/OS Direct Page	

GS/OS Direct Page Contents:

Offset	Parameter type	Size
$74	SCSI Manager SIB ptr.	Long
$78	Parameter list pointer	Long

Parameter List:

Offset	Parameter type	Size	Comments
$0000	SCSI Device type	Word	SCSI standard peripheral device type
$0002	InterComm vector	Long	Vector used for SCSI man to notify drivers of environment changes.
$0006	Claimed Device list ptr.	Long	Buffer must be $0704 bytes long.

Claimed Device List:

Offset	Parameter type	Size	Comments
$0000	Zero page address	Word	Start of zero page usable by scsi drivers
$0002	Total Device Count	Word	# devices matching SCSI device type requested
$0004	Device ID	Long	Bits 0-3 of low word contain Slot # 0-F device belongs to. Slots 0-7 are internal slots Slots 8-F are external slots High word contains the unit number assigned to the device by the Manager. Note the Device ID matches the slot and unit number definition for DIBs.
...	....	...	

Entry with offset $0004 will be repeated the number of times defined by field $0002 (Total Device count).

Errors:

$FE03 = Device list contains an ID which is not found by the Manager.

SCSI I/O Call   Call $0004

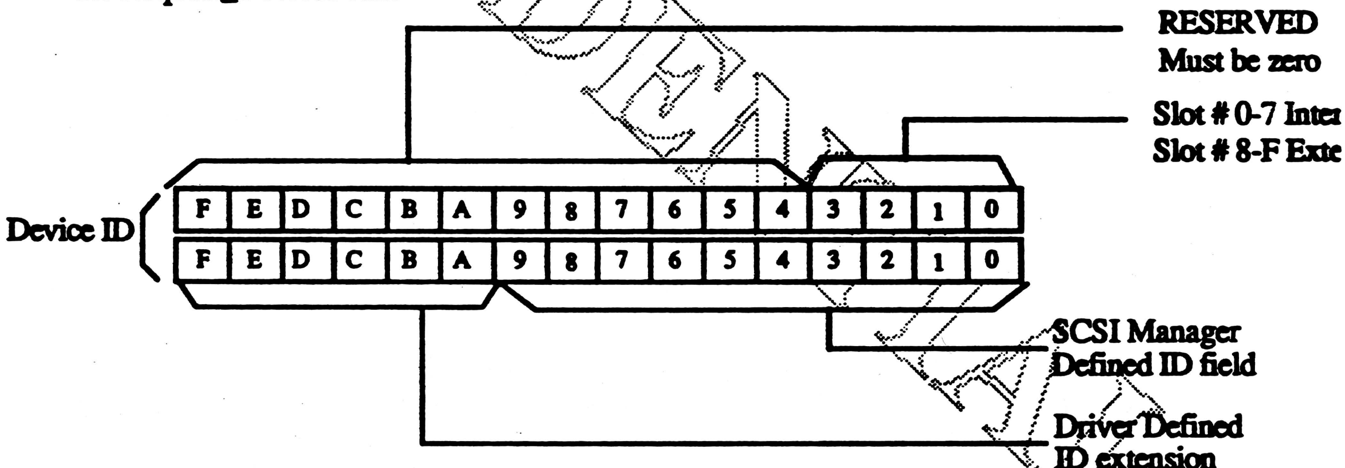
This call will be issued to the scsi manager for all calls to access SCSI target devices.

Device Call Parameter list:

Offset	Parameter type	Size	Comments
$0000	SCSI Device ID	Long	SCSI device number
$0004	Pblock Version	Word	Parameter block version (must be $0000)
$0006	Call Attributes	Word	Disconnect enable & message direction indicators
$0008	Call Time-out	Word	Number of 1/4 sec intervals call must complete within
$000A	Completion Vector	Long	Completion routine vector address
$000B	Ptr. to SCSI cmd	Long	Command list to be sent to a SCSI device
$0012	Ptr.to Buffer structure	Long	List of data chaining instructions
$0016	Req/Xfer byte count	Long	On Entry contains number of bytes sought On Exit contains number of bytes transferred
$001A	AutoSense buffer Ptr.	Long	Pointer to AutoSense buffer
$001E	Reserved	Long	Reserved for future expansion must be zero.
$0022	Reserved	Long	

Device ID:

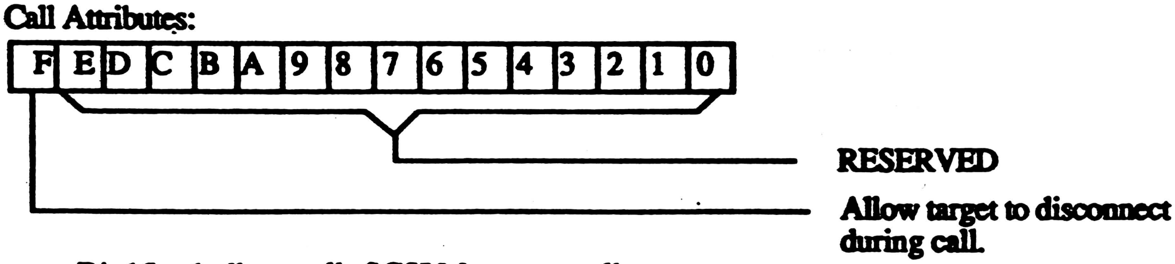
A long word field identifying the device sought. This is the ID given to the driver during the Request_Devices call.



Version:

Parameter block version number. Must be set to $0000 for this version of the manager.

Call Attributes:



Bit 15 = 1 allows tells SCSI Manager to allow target to disconnect if it chooses to do so during the call. If bit 15 = 0 then the SCSI Manager will not allow target to disconnect.

Call Time-out:

A word wide field defining the number of 1/4 second intervals the call must complete within.

Completion Vector:

The address of a routine that will be called via a JSL when a Async. SCSI command completes. There can only be one completion routine pending for each SCSI device. The Reselection interrupt comes into the SCSI Manager and it transfers control to the Completion routine address specified. The Completion will perform the function it needs to perform and return to the SCSI Manager through an RTL in full native mode.

SCSI Cmd:

Offset	Parameter type	Size	Comments
$0000	SCSI command	6,10, or 12	Actual SCSI Command

Buffer structure (Maximum of 32 Data chaining instructions)

Offset	Parameter type	Size	Comments
$0000	Data Chain Instruction	16 Bytes	(See DATA Chaining Section)
$0010	Data Chain Instruction	16 Bytes	
...	....		
$00XX	DCSTOP	16 Bytes	

Req/Xfer byte count:

This field contains the number of bytes requested in the call and will contain the number of bytes actually transferred on exit from the call.

AutoSense Buffer Pointer

Offset	Parameter type	Size	Comments
$0000	AutoSense Pointer	Long	Pointer to AutoSense buffer. Must be at least 256 bytes in size.

If the pointer = $00000000 then AutoSensing is disabled.
If the pointer is nonzero the SCSI Manager will issue a Request Sense call to the target and return the Sense data in the buffer when the call issued fails. The Request Sense buffer must be a minimum of one page in size. If the Request sense call fails then the error code will be in the range of $FE80-$FEFF. The low byte of the error code will contain the value of the SCSI status byte returned by the target device when the call failed which forced the Request Sense call to be issued.

Errors:
$FE02 = SCSI Check Condition from SCSI Device
$FE03 = Device ID not found
$FE05 = Parameter list
$FE08 = Device already busy.
$FE11 = Call Timeout
$FE80-$FEFF = Request Sense failed

Date: April 25, 1989

Author: Lou Infeld

Subject: System Loader Delta ERS

Document Version Number: 02:60

Revision History

02:60	(4/25/89)	Correction to Input description of InitialLoad2 Error $1101 added to UserShutDown
02:50	(2/23/89)	File Reference Number added to Pathname Table

Pathname Table

The **Pathname Table** is created by **System Loader** to remember the pathnames associated with each Load File it comes across. Note that the pathnames are fully expanded pathnames which are Type 1 strings (preceded by a word count rather than a byte count). At initial load, the **System Loader** creates the first entry in the **Pathname Table** from the pathname specified in the Initial Load function call. During the load, if the **System Loader** comes across a Pathname Segment (KIND=$04), it adds all the pathname entries to the **Pathname Table.** If Run Time Library Files are referenced during program execution, other Pathname Segments may be added.

Each entry in the **Pathname Table** is in the following format:

```
-----------------------------------
Next entry handle (4 bytes)
Previous entry handle (4 bytes)
UserID (2 bytes)
File Number (2 bytes)
File Date/Time (8 bytes)
Direct Page/Stack Address (2 bytes)
Direct Page/Stack Size (2 bytes)
Jump Table Segment Processed Flag (2 bytes)
Starting Address (4 bytes)
File Reference Number (2 bytes)
File Pathname (Pascal string)
-----------------------------------
```

where:

"Next entry handle" is the memory handle of the next entry in the **Pathname Table**. This handle is 0 in the last entry.

"Previous entry handle" is the memory handle of the previous entry in the **Pathname Table**. This handle is 0 in the first entry.

"UserID" is the UserID associated with this entry. In general, each Load File and each Run Time Library will have a different UserID and one entry in the **Pathname Table.** When a Run Time Library is first encountered during an Application execution, the **System Loader** will have a Run Time Library type of UserID assigned to it.

"File Number" is a number assigned by the **Linker** or **System Loader** for a specific Load File. File number 1 is reserved for the initial Load File.

The "File Date/Time" is the GS/OS directory item that the **Linker** retrieved during the link process. The **System Loader** will compare this value with the GS/OS directory of the Run Time Library File at run time. If they don't compare, the

System Loader will not load the requested Load Segment. This facility guarantees that the Run Time Library File used at link time is the <u>same</u> Run Time Library File loaded at execution time.

The "Direct Page/Stack" Address and Size is the information about the Direct Page and Stack buffer that was allocated during the Initial Load of this Load File (not applicable to Run Time Library Load Files). This allows the Restart function to resurrect an application without performing a Get File Info call on the Load File.

The "Jump Table Segment Processed Flag" indicates whether the Jump Table Segment (if any) in the Load File has been loaded yet. This flag will always be set for Initial Load Files but will be clear for Run Time Library Files. The first time a Segment in a Run Time Library File is requested, the **System Loader** will first load all its static load segments including the Jump Table Segment.

The "Starting Address" is the Starting Address of the Load File or Code Resource.

The "File Reference Number" is the GS/OS reference number associated with this file if the file is currently open. Otherwise, this number will be 0.

The "File Pathname" is the full pathname of this entry.

InitialLoad2 ($20)

Input:	UserID (2 bytes)	
	Input Address or parameter block (4 bytes)	
	Don't Use Special Memory Flag (2 bytes)	
	Input Type (2 bytes)	
Output:	UserID (2 bytes)	
	Starting Address (4 bytes)	
	Address of Direct Page/Stack buffer (2 bytes)	
	Size of Direct Page/Stack buffer (2 bytes)	
Errors:	$0000	- Operation succcessful
	$1102	- OMF Version error
	$1104	- File not Load File
	$1109	- SegNum out of sequence
	$110A	- Illegal load record found
	$110B	- Load Segment is foreign
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function is similar to InitialLoad except that three variations of the input information are available depending on the value of InputType.

If Input Type is 0, this function is exactly equivalent to the InitialLoad call.

If Input Type is 1, the Input Address points to a Load File Pathname which is in GS/OS string Type 1 format rather than Type 0. All this means is that the pathname string starts with a word count rather than a byte count.

If Input Type is 2, the Input Address points to a parameter which contains two parameters: a Memory Address (4 bytes) and a Length (2 bytes). These parameters specify where a Load File resides in memory and the **System Loader** will load it from memory rather than from a file. This input type is used by GS/OS at System Boot to load Load Files which were previously read into memory as binary images. In this mode, the **System Loader** does not make any GS/OS calls and can therefore be used when GS/OS is not in memory or has not yet been initialized.

If Input Type is 3, the Input Address points to an entry in the Pathname Table. The Pathname, UserID and File Number from the Pathname Table entry are used as input for the Initial Load. This entry is used by the Jump Table Load function to load all the static segments in a Run Time Library.

If Input Type is 4, the Input Address points to a parameter block which contains two parameters: a Memory Address (4 bytes) and a Length (2 bytes). These parameters specify where a Resource Code File resides in memory and the **System Loader** will load it from memory rather than from a file. This input type is used by the Resource Manager to relocate Code Resources that have been read into memory. The System Loader performs exactly the same function as a Memory Load (Input Type 2) except that

all the information about the Load Segments is purged from the **System Loader**'s tables.

UserShutDown ($12)

Input:	UserID (2 bytes)	
	Quit Flag (2 bytes)	
Output:	UserID (2 bytes)	
Errors:	$0000	- Operation succcessful
	$1101	- UserID not found
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function is called by the **Controlling Program** to close down an application which has just terminated. If the UserID specified is 0, the current UserID (USERID) is assumed.

The Quit Flag corresponds to the Quit Flag used in the GS/OS Quit call.

If the Quit Flag is 0, all Memory Blocks for the UserID are disposed and all the **System Loader**'s internal tables are purged of the UserID. The application can not be Restarted. The UserID is also removed from the system so that it can be reused.

if the Quit Flag is $8000, all Memory Blocks for the UserID are disposed and all the **System Loader**'s internal tables, except the Pathname Table, are purged of the UserID. The application can be reloaded but not Restarted because the pathname for the application is remembered.

If the Quit Flag is any other value, the memory blocks associated with the specified UserID (with Aux ID cleared) are processed as follows:

all memory blocks corresponding to Dynamic Load Segments are disposed
all memory blocks corresponding to Static Load Segments are made purgeable
all other memory blocks are purged

In addition, all dynamic segment entries in the **Memory Segment Table** and all entries in the **Jump Table List** for the specified UserID are removed. The application is now in a "zombie" state and can be resurrected by the **System Loader** very quickly because all the static segments are still in memory. However, as soon as any one static segment is purged by the **Memory Manager** for whatever reason, the **System Loader** must reload the application from its original Load File.

Date: April 25, 1989

Author: Lou Infeld

Subject: System Loader ERS

Document Version Number: 02:60

Revision History

02:60	(4/25/89)	Correction to Input description of InitialLoad2 Error $1101 added to UserShutDown
02:50	(2/23/89)	File Reference Number added to Pathname Table
02:40	(11/18/88)	Input Type 4 in the InitialLoad2 function changed Load Code Resource and Unload Code Resource functions removed Code Resource information removed from Pathname Table
02:30	(10/17/88)	Input Type 4 added to the InitialLoad2 function Size of parameters for Type 2 InitialLoad2 changed Load Code Resource and Unload Code Resource functions added Starting Address added to Pathname Table Code Resource information added to Pathname Table ENTRY field of Load Segment header is used to calculate the starting address for the Initial Load function
02:20	(8/4/88)	Rename Pathname Function added Memory Manager interface discussion cleaned up a little Example of how the System Loader finds Segments in LoadSegNum function was removed
02:10	(4/19/88)	Error $1105 removed Discussion of Volume not found changed in "General" section Description of pathnames in Pathname Table changed File Date and File Time fields in the Pathname Table changed Description of the Jump Table Segment Processed Flag in the Pathname Table changed Another Input Type added to the InitialLoad2 function
02:00	(10/16/87)	Changes made to support GS/OS Pathname Table now contains Type 1 strings rather than Type 0 Pathname Table now contains completely expanded pathnames without exceptions InitialLoad2, GetUserID2 and LGetPathname2 added All other functions with pathname inputs and outputs assume Type 0 strings Functions that check the Load File Type now check it after the file is opened rather than before In the Restart function, the Aux ID in the input UserID is no longer ignored In the UserShutDown, the AuxID is not set to 0 References to Cortland were removed A warning added to LGetPathname

		Error $1103 added
01:30	(3/5/87)	Description of automatic handling of Off-Line volumes added to General description
		Description of Cleanup Routine changed
		Description of UserShutDown changed
		Description of Restart changed
		Description of Memory Manager Interface changed
01:20	(12/17/86)	Jump Table Segment Processed Flag added to Pathname Table
		Description of initial loading of Direct Page/Stack Segments changed
		Support of Reload and Initialization Segments added to Restart
		Example of finding a Load Segment changed
		Output from LoadSegName changed and support for Jump Table Segment Processed Flag added
		Description of UserShutDown changed
01:10	(11/14/86)	Changes made in support of OMF Version 2
		Function name changes to correspond to macros
		OMF Version check logic added
		Error $1102 added
		UserID removed at UserShutdown in certain cases
01:00	(9/9/86)	Definition of System Loader version 1.0

Table of Contents

History

The Apple // under ProDOS has a very basic System Loader. It is the part of the boot code that searches the boot disk for the first System file (any file of type $FF whose name ends with ".SYSTEM") and loads it into location $2000. If a System program wants to load another System program, it has to do all the work by making ProDOS calls.

Some programming environments such as Apple // Pascal and AppleSoft Basic provide loaders for programs running under them. The AppleSoft loader loads either System files, Basic files or binary code files. All these files are loaded either at a fixed address in memory or at an address specified in the file.

Since the Apple //GS has a large amount of "clean" memory, a more dynamic load facility is needed. Programs should be able to be loaded anywhere that is available in memory. The burden of determining where to load a program should be on a loader and not on the applications programmer. Also programs should be able to be broken into smaller program segments which can be loaded independently.

Therefore on the Apple //GS, there is a relocating System Loader. Files generated by the Linker are loadable by the System Loader. The System Loader provides a very powerful and flexible facility that was not available on the Apple //.

Overview

The System Loader will load programs or program segments by first calling the Memory Manager to find available memory. It will perform relocation during the load as necessary and will load each segment independently. Therefore, a large program can be broken up into smaller program segments each of which is loaded at separate locations in memory. Program segments can also be loaded dynamically as they are referenced rather than at program boot time. Additionally, the System Loader can be called by the program itself to load and unload program (or data) segments.

Definitions

The **Linker** is the program that combines files generated by compilers and assemblers, resolves all symbolic references and generates a file that can be loaded into memory and executed.

The **System Loader** is the part of the Operating System that reads the files generated by the **Linker** and loads them into memory (performing relocation if necessary).

Object Files are the output from an assembler or compiler and the input to the **Linker**.

Library Files are files containing general program Segments that the **Linker** can search.

Load Files are the output of the **Linker** and contain memory images which the **System Loader** will load into memory. Shell Load Files and Startup Load Files are special Load Files used by the Shell and GS/OS respectively.

Run Time Library Files are Load Files that contain general program Segments which can be loaded as needed by the **System Loader** and shared between applications.

Object Module Format is the general format used in Object Files, Library Files and Load Files.

An **OMF File** is a file in Object Module Format (i.e. an Object File, Library File or Load File).

A **Segment** is a individual component of an OMF file. Each file contains one or more Segments.

A **Code Segment** is a Segment in an Object File that contains program code.

A **Data Segment** is a Segment in an Object File that contains program data.

A **Load Segment** is a Segment in a Load File.

The **Controlling Program** is the program that requests the **System Loader** to initially load and run other programs and is responsible for shutting these programs down when they exit. A Finder is an example of a Controlling Program.

General

The **System Loader** processes files which conform to the Apple //GS definition of a Load File (see Apple //GS ProDOS16 Reference). A Load File consists of Load Segments, each of which can be loaded independently. The Load Segments are numbered sequentially from 1.

Certain Load Segments are Static Load Segments. These Segments are meant to be loaded into memory at initial program load time and must stay in memory until program completion.

Another general type of Load Segment is the Dynamic Load Segment. These Segments are not loaded at boot time. They are loaded dynamically during program execution. This can happen automatically by means of the **Jump Table** mechanism or manually at the specific request of the application. When these Segments are not being referenced, they can be purged by the **Memory Manager.**

There are several other attributes that Load Segments can have (see OMF ERS for a complete list of attributes).

There are several special types of Load Segments. The **Jump Table Segment** (KIND=$02), when loaded into memory, provides a mechanism whereby Segments in memory can trigger the loading of other Segments not yet in memory.

The **Pathname Segment** (KIND=$04) contains information about the Load Files that are referenced.

The **Initialization Segment** (KIND=$10) is used for code that is to be executed before all the rest of the Load Segments are loaded.

The **Direct Page/Stack Segment** (KIND=$12) defines the application's Direct Page and stack requirements. This segment will be loaded into Bank 0 and its starting address and length are passed to the **Controlling Program** who will set the Direct Register and Stack Pointer to the start and end of this segment before transferring control to the program.

During the initial load, the **System Loader** has all the information needed to resolve all inter-segment references between the Static Load Segments. But during the dynamic loading of Dynamic Load Segments, it can only resolve references in the Dynamic Load Segment to the already loaded Static Load Segments. Therefore, the general rule is that Static Segments can be referenced by any type of segment but Dynamic Segments can only be referenced through JSL calls through the **Jump Table**.

If the **System Loader** is called to perform the initial load of a program, it will load all the Static Load Segments and the Segment Jump and Pathname Tables (if they exist). A RAM based **Memory Segment Table** will be constructed during this process.

If the **System Loader** references a file on a volume that is not mounted, GS/OS will

either return with a Volume not found error ($45) or will display a mount message depending on the state of the System Preferences at the time of the GS/OS call. If a mount message is displayed, GS/OS handles the user interface and only returns control to the **System Loader** when the I/O operation is completed or the user has canceled the request for the mount. For all user-callable **System Loader** functions, the System Preference is controlled by the user. For the internal Jump Table Load function, the **System Loader** sets the System Preferences to display mount messages and then restores them to their original states.

Memory Manager Interface

The **System Loader** and the **Memory Manager** work closely together.

When the **System Loader** loads Static Segments, it calls the **Memory Manager** to allocate corresponding memory blocks which are marked as unpurgeable and unmovable. Dynamic Segments are marked as purgeable but locked. Position Independent Segments are marked as moveable.

When the **System Loader** unloads a specific segment, it calls the **Memory Manager** to purge the corresponding memory blocks. However, if the **Controlling Program** wishes to unload all segments associated with a UserID (application shut-down), it calls the **System Loader** Application Shutdown function which calls the **Memory Manager** to first purge all Dynamic Segments for the UserID and then make all the Static Segments purgeable. The purpose of this is to keep an application in memory, if possible, in case it needs to be re-loaded in the near future. This will greatly speed up a Finder or Switcher. The complication occurs when the **Memory Manager** has to actually purge one of the segments of a User. Now the application is now incomplete in memory and can therefore not be restarted.

If many "incomplete" applications are in memory, the system may get bogged down with NIL memory handles. To avoid this situation, the **System Loader** will attempt to dispose all NIL memory handles it knows about before every Initial Load or Restart.

The relationship between a Load Segment in a Load File and the corresponding memory block is very close. The average Load Segment will be loaded into a memory block having the attributes:

Locked
Fixed
Purge Level=0 (for Static)
Purge Level=1 (for Dynamic)

Depending on the ORG, KIND, BANKSIZE and ALIGN fields in the Segment Header, other memory attributes will be used:

if ORG>0, the "Fixed Address" attribute is set.
if BANKSIZE=$10000, the "May not cross bank boundary" attribute is set.
if 0<BANKSIZE<$10000 then use Align factor=MAX(BANKSIZE,ALIGN)
    otherwise use Align factor=ALIGN:
if 0<Align Factor<=$100, the "Page Aligned" attribute is set.
if Align Factor>$100, Bank Alignment is forced (not an attribute).
if the Position Independent bit of KIND is set, the "Fixed" attribute is
    removed.
if the Absolute Bank bit of KIND is set, the "Fixed Address" attribute is
    removed and the "Fixed Bank" attribute is set.
if KIND indicates Direct Page/Stack Segment, the "Fixed Bank" and "Page
    Aligned" attributes are set.

A memory block can be made purgeable ("unloaded") by a call to the **System Loader**. However, the other attributes must be changed through **Memory Manager** calls. Since the Memory Handle for a memory block is stored in the **Memory Segment Table, Memory Manager** information is accessible. Other memory block information that may be useful to a program are:

Start location
Size of segment
UserID
Purge Level (0 - UnPurgeable
1 - Least Purgeable
3 - Most Purgeable)

Also, if the Memory Handle is NIL (i.e the Memory Address is 0), the memory block has been purged.

Restrictions

The Object Module Format and the Linker have general capabilities above what is needed or desired for the Apple //GS computer. The **System Loader**, on the other hand, is designed specifically for the Apple //GS computer. Therefore, there are certain abilities that are not supported or are restricted. This section will list these differences.

The NUMSEX field of the Segment Header must be 0.
The NUMLEN field of the Segment Header must be 4.
The BANKSIZE field of the Segment Header must be <=$10000.
The ALIGN field of the Segment Header must be <=$10000.

If any of the above is not true, the **System Loader** will return with a "Segment is foreign" error ($110B). The BANKSIZE and ALIGN restrictions will be enforced by the **Linker** and should not make it to the Load File.

ALIGN and BANKSIZE can be any multiple of 2. The **Memory Manager**, and therefore the **System Loader**, can not handle so general a requirement. The **Memory Manager** can currently only be told that a memory block be page aligned or not cross a bank boundary. The **Memory Manager** may handle Bank Alignment in the near future. All is not lost, however, because the **System Loader** will fulfill the general requirements in the following, somewhat inefficient, way:

Any value of BANKSIZE other than 0 and $10000 will result in a memory block that is either page aligned (if BANKSIZE<=$100) or bank aligned (if BANKSIZE>$100). Since the **Linker** will make sure that the segment is smaller than BANKSIZE, the requirement that the segment not extend past the BANKSIZE boundary will be met (there will be wasted space in the memory block however).

Any value of ALIGN will be bumped to either page alignment or bank alignment.

If there is a BANKSIZE other than 0 and $10000 and a non-zero ALIGN, the maximum of the two will be used to determine the alignment to be used.

Data Structures

Globals

SEGTBL	--	Absolute address of Memory Segment Table
JMPTBL	--	Absolute address of Jump Table List
PATHTBL	--	Absolute address of Pathname Table
USERID	--	UserID of current application

Memory Segment Table

The **Memory Segment Table** is a linked list. Each entry corresponds to one memory block known to the **System Loader**. These memory blocks were the result of loading Load Segments from a Load File. The format of each entry in the **Memory Segment Table** is:

Next entry handle	-- 4 bytes
Previous entry handle	-- 4 bytes
UserID	-- 2 bytes
Memory Handle	-- 4 bytes
Load File Number	-- 2 byte
Load Segment Number	-- 2 bytes
Load Segment Kind	-- 2 bytes

where:

"Next entry handle" is the memory handle of the next entry in the Memory Segment Table. This handle is 0 in the last entry.

"Previous entry handle" is the memory handle of the previous entry in the Memory Segment Table. This handle is 0 in the first entry.

"UserID" is the UserID associated with this segment. It is needed in case the "Memory Handle" is NIL and the UserID can therefore not be determined directly from the Memory Manager.

"Memory Handle" is the handle of the memory block obtained from the **Memory Manager**. More information about the segment is available through this handle (e.g. UserID, Purge Priority).

"Load File Number" corresponds to the Load File or Run Time Library File from which the segment was obtained. If this number is 1, this segment is in the initial Load File.

"Load Segment Number" is the segment number of the Load Segment in the Load File.

"Load Segment Kind" is the KIND field from the Segment Header of this segment.

Jump Table List

The **Jump Table List** (or **Jump Table**) is the mechanism that allows programs to reference segments that are loaded into memory only when they are needed. The **Jump Table** is a linked list containing the UserID and Handle to each **Jump Table Segment** (KIND=$02) that the **System Loader** has encountered. Any Load File and Run Time Library File may contain a **Jump Table Segment.** The format of each entry in the **Jump Table List** is:

Next entry handle	-- 4 bytes
Previous entry handle	-- 4 bytes
UserID	-- 2 bytes
Memory Handle	-- 4 bytes

where:

"Next entry handle" is the memory handle of the next entry in the Jump Table List. This handle is 0 in the last entry.

"Previous entry handle" is the memory handle of the previous entry in the Jump Table List. This handle is 0 in the first entry.

"UserID" is the UserID associated with this Jump Table Segment.

"Memory Handle" is the handle of the memory block associated with this Jump Table Segment.

When the **Linker** encounters a JSL to an external Dynamic Segment, it creates an entry in the **Jump Table Segment**. It then links the JSL to the **Jump Table Segment** entry it just created. The format of this entry in the **Jump Table Segment** is:

UserID (2 bytes)
Load File Number (2 bytes)
Load Segment Number (2 bytes)
Load Segment Offset (4 bytes)
jsl Jump Table Load Function

where the Load File Number, Segment Number and Offset refer to the location of the external reference. The rest of the entry is a call to the **System Loader** Jump Table Load function. The UserID and the actual address of the **System Loader** function will be patched by the **System Loader** during Initial Load. This format is considered the "unloaded" state of the entry.

When the JSL instruction actually executes, control is transferred to the **Jump Table** entry which in turn transfers to the **System Loader**. The **System Loader** extracts the segment information from the **Jump Table** entry, the file information from the **Pathname Table** and loads the Dynamic Segment, changes the entry in the **Jump**

Table to its "loaded" state and transfers to the location in the just loaded segment. Typically, the location in the loaded segment is a subroutine and when it exits with a RTL, control is eventually transferred to the location following the original JSL instruction.

The loaded state of a **Jump Table** entry is very similar to the unloaded state except that the JSL to the **System Loader** Jump Table Load function is replaced by a JML to the external reference. A typical loaded entry would look like this:

UserID (2 bytes)
Load File Number (2 bytes)
Load Segment Number (2 bytes)
Load Segment Offset (4 bytes)
jml external reference

Pathname Table

The **Pathname Table** is created by **System Loader** to remember the pathnames associated with each Load File it comes across. Note that the pathnames are fully expanded pathnames which are Type 1 strings (preceded by a word count rather than a byte count). At initial load, the **System Loader** creates the first entry in the **Pathname Table** from the pathname specified in the Initial Load function call. During the load, if the **System Loader** comes across a Pathname Segment (KIND=$04), it adds all the pathname entries to the **Pathname Table**. If Run Time Library Files are referenced during program execution, other Pathname Segments may be added.

Each entry in the **Pathname Table** is in the following format:

Next entry handle (4 bytes)
Previous entry handle (4 bytes)
UserID (2 bytes)
File Number (2 bytes)
File Date/Time (8 bytes)
Direct Page/Stack Address (2 bytes)
Direct Page/Stack Size (2 bytes)
Jump Table Segment Processed Flag (2 bytes)
Starting Address (4 bytes)
File Reference Number (2 bytes)
File Pathname (Pascal string)
----

where:

"Next entry handle" is the memory handle of the next entry in the **Pathname Table**. This handle is 0 in the last entry.

"Previous entry handle" is the memory handle of the previous entry in the **Pathname**

Table. This handle is 0 in the first entry.

"UserID" is the UserID associated with this entry. In general, each Load File and each Run Time Library will have a different UserID and one entry in the **Pathname Table**. When a Run Time Library is first encountered during an Application execution, the **System Loader** will have a Run Time Library type of UserID assigned to it.

"File Number" is a number assigned by the **Linker** or **System Loader** for a specific Load File. File number 1 is reserved for the initial Load File.

The "File Date/Time" is the GS/OS directory item that the **Linker** retrieved during the link process. The **System Loader** will compare this value with the GS/OS directory of the Run Time Library File at run time. If they don't compare, the **System Loader** will not load the requested Load Segment. This facility guarantees that the Run Time Library File used at link time is the same Run Time Library File loaded at execution time.

The "Direct Page/Stack" Address and Size is the information about the Direct Page and Stack buffer that was allocated during the Initial Load of this Load File (not applicable to Run Time Library Load Files). This allows the Restart function to resurrect an application without performing a Get File Info call on the Load File.

The "Jump Table Segment Processed Flag" indicates whether the Jump Table Segment (if any) in the Load File has been loaded yet. This flag will always be set for Initial Load Files but will be clear for Run Time Library Files. The first time a Segment in a Run Time Library File is requested, the **System Loader** will first load all its static load segments including the Jump Table Segment.

The "Starting Address" is the Starting Address of the Load File or Code Resource.

The "File Reference Number" is the GS/OS reference number associated with this file if the file is currently open. Otherwise, this number will be 0.

The "File Pathname" is the full pathname of this entry.

Mark List

The **Mark List** is created by **System Loader** to remember the file locations of the relocation dictionary of each Load Segment. The format of the **Mark List** is:

```
-----------------------------------
Next Available Slot (4 bytes)
End of Table (4 bytes)
Segment Number (2 bytes)
File Mark (4 bytes)
Segment Number (2 bytes)
File Mark (4 bytes)
Segment Number (2 bytes)
File Mark (4 bytes)
...
...
Segment Number (2 bytes)
File Mark (4 bytes)
-----------------------------------
```

where:

"Next Available Slot" is the relative offset of the next empty entry in the **Mark List**.

"End of List" is the relative offset to the end of the **Mark List.**

The **Mark List** is initially large enough for 100 Marks and grows larger as needed.

Functions

General

Since the **System Loader** is a Apple //GS Tool, its functions are called by making calls through the Apple //GS Tool mechanism. The calling sequence for **System Loader** functions is the standard Tool calling sequence. Space for the output parameter (if any) is pushed on the stack followed by each input parameter ***in the order specified in the function description***. This is followed by:

```
ldx #$11+FuncNum|8
jsl Dispatcher
```

where "FuncNum" is the **System Loader** function number and the "$11" is the Tool Number for the **System Loader**. Upon return, the A register will contain the status and the Carry will be set if an error occurred. If there is output, each output parameter must be pulled off the stack ***in the order specified in the function description***.

The Jump Table Load function does not use the above calling sequence. It can not be called by an application directly but is called indirectly by a Jump Table entry. In this case the absolute address of the function is patched by the **System Loader**.

LoaderInitialization ($01)

Input:	none
Output:	none
Errors:	none

This function will initialize the System Loader. It should only be called at system initialization time. All System Loader tables are cleared and no assumptions are made about the current or previous state of the system.

LoaderStartup ($02)

Input:	none
Output:	none
Errors:	none

This function does nothing and need not be called.

LoaderShutDown ($03)

Input:	none
Output:	none
Errors:	none

This function does nothing and need not be called.

LoaderVersion ($04)

Input:	none
Output:	Loader Version (2 bytes)
Errors:	none

This function will return the Version Number of the System Loader.

LoaderReset ($05)

Input:	none
Output:	none
Errors:	none

This function does nothing and need not be called.

LoaderStatus ($06)

Input:	none
Output:	Status (TRUE or FALSE 2 bytes)
Errors:	none

This function will always return TRUE since the **System Loader** will always be in the initialized state.

InitialLoad ($09)

Input:	UserID (2 bytes)	
	Address of Load File Pathname (4 bytes)	
	Don't Use Special Memory Flag (2 bytes)	
Output:	UserID (2 bytes)	
	Starting Address (4 bytes)	
	Address of Direct Page/Stack buffer (2 bytes)	
	Size of Direct Page/Stack buffer (2 bytes)	
Errors:	$0000	- Operation succcessful
	$1102	- OMF Version error
	$1104	- File not Load File
	$1109	- SegNum out of sequence
	$110A	- Illegal load record found
	$110B	- Load Segment is foreign
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

A **Controlling Program** (such as GS/OS, Basic, Switcher, etc.) will call the **System Loader** to perform an "Initial Load".

If a complete UserID is specified, the **System Loader** will use that when allocating memory for the Load Segments. If the Main ID portion of the UserID is 0, a new UserID is obtained from the UserID Manager based on the Type portion of the UserID. If the Type portion is 0, an Application type UserID is requested from the UserID Manager.

If the Don't Use Special Memory Flag is TRUE (i.e. not 0), the **System Loader** will **NOT** load any static load segments into Special Memory. However, dynamic load segments will be loaded into any memory.

GS/OS is called to open the specified Load File using the input pathname. Note that the input pathname is a Type 0 string (Pascal string). If any GS/OS errors occurred or if the file is not a Load File type ($B3-$BE), the **System Loader** will return the appropriate error.

If the Load File was successfully opened, the **System Loader,** adds the Load File information to the **Pathname Table**, and calls the Load Segment by Number function for each Static Load Segment in the Load File.

If an Initialization Segment (KIND=$10) is loaded, the **System Loader** will immediately transfer control to that segment in memory. When the **System Loader** regains control, the rest of the static segments are loaded normally.

If the Direct Page/Stack Segment (KIND=$12) is loaded, its starting address and length are returned as output.

If any of the static segments could not be loaded, the **System Loader** will abort the load and return the error.

After all the Static Load Segments have been loaded, return is made to the **Controlling Program** with the entry address of the first Load Segment (not an Initialization Segment) of File Number 1. The entry address is the address in memory of the Load Segment plus the ENTRY field in the Load Segment Header. Note that the **Controlling Program** is responsible for setting up the stack and Direct Page registers and actually transferring control to the loaded program.

InitialLoad2 ($20)

Input:	UserID (2 bytes) Input Address or parameter block (4 bytes) Don't Use Special Memory Flag (2 bytes) Input Type (2 bytes)	
Output:	UserID (2 bytes) Starting Address (4 bytes) Address of Direct Page/Stack buffer (2 bytes) Size of Direct Page/Stack buffer (2 bytes)	
Errors:	$0000	- Operation succcessful
	$1102	- OMF Version error
	$1104	- File not Load File
	$1109	- SegNum out of sequence
	$110A	- Illegal load record found
	$110B	- Load Segment is foreign
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function is similar to InitialLoad except that three variations of the input information are available depending on the value of InputType.

If Input Type is 0, this function is exactly equivalent to the InitialLoad call.

If Input Type is 1, the Input Address points to a Load File Pathname which is in GS/OS string Type 1 format rather than Type 0. All this means is that the pathname string starts with a word count rather than a byte count.

If Input Type is 2, the Input Address points to a parameter block which contains two parameters: a Memory Address (4 bytes) and a Length (2 bytes). These parameters specify where a Load File resides in memory and the **System Loader** will load it from memory rather than from a file. This input type is used by GS/OS at System Boot to load Load Files which were previously read into memory as binary images. In this mode, the **System Loader** does not make any GS/OS calls and can therefore be used when GS/OS is not in memory or has not yet been initialized.

If Input Type is 3, the Input Address points to an entry in the Pathname Table. The Pathname, UserID and File Number from the Pathname Table entry are used as input for the Initial Load. This entry is used by the Jump Table Load function to load all the static segments in a Run Time Library.

If Input Type is 4, the Input Address points to a parameter block which contains two parameters: a Memory Address (4 bytes) and a Length (2 bytes). These parameters specify where a Resource Code File resides in memory and the **System Loader** will load it from memory rather than from a file. This input type is used by the Resource Manager to relocate Code Resources that have been read into memory. The System Loader performs exactly the same function as a Memory Load (Input Type 2) except that all the information about the Load Segments is purged from the **System Loader**'s tables.

Restart ($0A)

Input:	UserID (2 bytes)	
Output:	UserID (2 bytes)	
	Starting Address (4 bytes)	
	Address of Direct Page/Stack buffer (2 bytes)	
	Size of Direct Page/Stack buffer (2 bytes)	
Errors:	$0000	- Operation succcessful
	$1101	- Application not found
	$1108	- UserID error
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

A **Controlling Program** (such as GS/OS, Basic, Switcher, etc.) can call the **System Loader** to perform a "restart" of an application still in memory. Only software that is "reentrant" can be successfully restarted. For a program to be "reentrant", it must initialize its variables and not assume that they will be preset at Load time. A Reload Segment can be used for initializing data because it is reloaded from the file during a Restart. The **Controlling Program** must determine whether a given program can be restarted.

An existing UserID must be specified, otherwise the **System Loader** will return error $1108. If the UserID is not known to the **System Loader**, error $1101 will be returned.

Applications can be "restarted" only if all the segments in the Memory Segment table with the specified UserID are in memory. Note these segments are the application's static segments. If this is the case, the **System Loader** resurrects the application by calling the Memory Manager to lock and make all its segments unpurgeable. The UserID and the starting address obtained from the first segment are returned as well as the Direct Page/Stack information from the Pathname Table. After all the static segments are resurrected, the **System Loader** looks for Initialization and Reload Segments and executes the former and reloads the latter.

If there is a Pathname Table entry for the UserID but not all the segments are in memory, the **System Loader** will first do a UserShutdown which will purge the UserID from all its tables and then perform an Initial Load from the original Load File.

LoadSegNum (Load Segment by Number) ($0B)

Input:	UserID (2 bytes)	
	Load File Number (2 bytes)	
	Load Segment Number (2 bytes)	
Output:	Address of segment (4 bytes)	
Errors:	$0000	- Operation succcessful
	$1101	- Segment not found
	$1102	- OMF Version error
	$1104	- File not Load File
	$1107	- File Version error
	$1109	- SegNum out of sequence
	$110A	- Illegal load record found
	$110B	- Segment is foreign
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function will load a specific Load Segment into memory. This is the workhorse function of the **System Loader**. Normally, a program will call this function to manually load a Dynamic Load Segment. If a program calls this function to load a Static Load Segment, the **System Loader** will not patch any existing references to the newly loaded segment.

First the **Memory Segment Table** is searched to see if there is an entry for the requested Load Segment. If there is already an entry, the handle to the memory block is checked to verify it is still in memory. If it is still in memory, this function does nothing further and returns without an error. If the memory block has been purged, the **Memory Segment Table** entry is deleted.

Next the "Load File Number" is looked up in the **Pathname Table** to get the Load File pathname.

Next the Load File type is checked. If it is not a Load File (types $B3-$BE), error $1104 is returned.

Next the Load File's "last_mod" value is compared to File Date and File Time values in the **Pathname Table**. If these values do not match, error $1107 is returned. This indicates that the Run Time Library File at the specified pathname is not the Run Time Library File that was scanned when the application was linked together.

GS/OS is then called to open the specified Load File. If GS/OS has a problem, its error code is returned.

Next the Load File is searched for a Load Segment corresponding to the specified "Load Segment Number". If there is no segment corresponding to the "Load Segment Number", error $1101 is returned. If the VERSION field contains a value which is not supported by the **System Loader**, error $1102 is returned. If the SEGNUM field does not correspond to the "Load Segment Number", error $1109 is returned. If the NUMSEX and NUMLEN fields are not "0" and "4", error $110B is returned.

If the Load Segment is found and its Segment Header is correct, a memory block is requested from the **Memory Manager** of size specified in the LENGTH field in the Segment Header. If the ORG field in the Segment Header is not 0, a memory block starting at that address is requested. Other attributes are set according to Segment Header fields (see Memory Manager Interface section).

If the UserID specified is not 0, it is used as the UserID of the memory block. If the UserID specified is 0, the memory block will be marked as belonging to the UserID of the current User (in USERID).

If the requested memory is not available, the **Memory Manager** and the **System Loader** will try several techniques to free up memory:

The **Memory Manager** will purge memory blocks that are marked purgeable

The **Memory Manager** will move moveable segments to enlarge contiguous memory

The **System Loader** will call its Cleanup routine to free its own unused internal memory

If all these techniques fail, the **System Loader** will return with the last **Memory Manager** error.

Once enough memory is available, the Load Segment is loaded into memory and the relocation dictionary (if any) is processed. Note only the following Object Module Format records are supported by the **System Loader:**

LCONST	($F2)
DS	($F1)
RELOC	($E2)
INTERSEG	($E3)
cRELOC	($F5)
cINTERSEG	($F6)
SUPER	($F7)
END	($00)

Any other records encountered will result in a $110A error.

A new entry is added to the **Memory Segment Table**.

Finally, the **System Loader** returns with the Memory Handle of the memory block.

Note that since Load Segments in a Load File are numbered sequentially starting at 1, to find Load Segment 5, the **System Loader** must scan through the first 4 Load Segments before finding Load Segment 5. Each Load Segment Header must be processed because Load Segments as well as Load Segment Header are variable length.

UnloadSegNum (Unload Segment by Number) ($0C)

Input:	UserID (2 bytes)	
	Load File Number (2 bytes)	
	Load Segment Number (2 bytes)	
Output:	none	
Errors:	$0000	- Operation succcessful
	$1101	- Segment not found
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function will unload a specific Load Segment that is currently in memory.

The **System Loader** searches the **Memory Segment Table** for the "Load File Number" and "Load Segment Number". If there is no such entry, error $1101 is returned.

Next the **Memory Manager** is called to make the memory block purgeable using the Memory Handle in the table entry.

All entries in the **Jump Table** referencing the unloaded segment are changed to their "unloaded" states.

If the input UserID is 0, the UserID of the current user (in USERID) is assumed.

If both the Load File Number and the Load Segment Number are specified, the specific Load Segment is made purgeable whether it is static or dynamic. Note , if a static segment is unloaded, the application can not be ReStarted. If either input is 0, only dynamic segments will be made purgeable.

If the input Load Segment Number is 0, all dynamic segments in the specified Load File are unloaded.

If the input Load File Number is 0, all dynamic segments for the UserID are unloaded.

LoadSegName (Load Segment by Name) ($0D)

Input:	UserID (2 bytes)	
	Address of Load File Name (4 bytes)	
	Address of Load Segment Name (4 bytes)	
Output:	Address of segment (4 bytes)	
	UserID (2 bytes)	
	Load File Number	
	Load Segment Number	
Errors:	$0000	- Operation succcessful
	$1101	- Segment not found
	$1104	- File not Load File
	$1107	- File Version error
	$1109	- SegNum out of sequence
	$110A	- Illegal load record found
	$110B	- Load Segment is foreign
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function will load a named Load Segment into memory.

Note that the input pathname is a Type 0 string (Pascal string).

GS/OS is called to open the specified Load File. If GS/OS has a problem, its error code is returned. If the file is not a Load File (types $B3-$BE), error $1104 is returned.

Next the Load File is searched for a Load Segment corresponding to the specified "Load Segment Name". If there is no segment with Segment Name requested, error $1101 is returned.

Now that the **System Loader** has located the requested Load Segment (and knows the Load Segment Number), it checks the **Pathname Table** to see whether the Load File is represented. If so, it uses the File Number from the table. Otherwise, the **System Loader** adds a new entry to the **Pathname Table** with an unused File Number. If the Jump Table Segment Processed Flag in the **Pathname Table** is clear, the **System Loader** loads the Jump Table Segment (if any) from the Load File and sets the Flag.

Next the **System Loader** attempts to load this Load Segment by calling the Load Segment by Number function. If the Load Segment by Number function returns an error, the Load Segment by Name function, in turn, returns this error. If the Load Segment by Number function is successful, the Load Segment by Name function returns the Load File Number, the Load Segment Number and the Memory Address of the segment in memory.

UnloadSeg ($0E)

Input:	Address in Segment (4 bytes)	
Output:	UserID (2 bytes)	
	Load File Number (2 bytes)	
	Load Segment Number (2 bytes)	
Errors:	$0000	- Operation succcessful
	$1101	- Segment not found
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function will unload the Load Segment which contains the specified address.

The **Memory Manager** is called to locate the memory block containing the specified address. If no memory block contains the address, error $1101 is returned. The UserID associated with the Handle of the memory block returned by the **Memory Manager** is extracted (from the **Memory Manager**'s internal table). The **Memory Segment Table** is scanned looking for the UserID and Handle. If an entry is not found, error $1101 is returned.

If the entry in the **Memory Segment Table** is for a Jump Table Segment, the specified address should be pointing to the **Jump Table** entry for a dynamic segment reference. The Load File Number and Segment Number of the **Jump Table** entry are extracted.

If the entry in the **Memory Segment Table** is not for a Jump Table Segment, the Load File Number and Segment Number of the **Memory Segment Table** entry are extracted.

The UnloadSegNum function is now called to actually unload the segment. The outputs of this function can be used as input to other **System Loader** functions.

GetLoadSegInfo ($0F)

Input:	UserID (2 bytes)	
	Load File Number (2 bytes)	
	Load Segment Number (2 bytes)	
	Address of User Buffer (4 bytes)	
Output:	filled User Buffer	
Errors:	$0000	- Operation succcessful
	$1101	- Entry not found
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function will return the Memory Segment Table entry corresponding to the specified Load Segment.

The Memory Segment Table is searched for the specified entry. If the entry is not found, error $1101 is returned. If the entry is found, the contents except for the link pointers are moved into the User Buffer.

GetUserID ($10)

Input:	Address of Pathname (4 bytes)	
Output:	UserID (2 bytes)	
Errors:	$0000	- Operation succcessful
	$1101	- Entry not found
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function will search the Pathname Table for the specified Pathname. Note that the input pathname is a Type 0 string (Pascal string). The Pathname is first expanded to a full pathname before the search. If a match is found, the corresponding UserID is returned. A **Controlling Program** can use this function to determine whether to perform a Restart of an application or an Initial Load.

GetUserID2 ($21)

Input:	Address of Pathname (4 bytes)	
Output:	UserID (2 bytes)	
Errors:	$0000	- Operation succcessful
	$1101	- Entry not found
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function is identical to GetUserID except that the input Pathname is assumed to be a Type 1 string rather than a Type 0 string.

LGetPathname ($11)

Input:	UserID (2 bytes)	
	File Number (2 bytes)	
Output:	Address of Pathname (4 bytes)	
Errors:	$0000	- Operation succcessful
	$1101	- Entry not found
	$1103	- Pathname error
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function will search the Pathname Table for the specified UserID and File Number. If a match is found, the address of the Pathname in the Pathname Table is returned. Note that the output pathname is a Type 0 string (Pascal string). GS/OS uses this call to get the pathname of an existing application so that it can set the Application prefix before restarting it. Note that the output address is within a **System Loader** internal data structure and nothing should be written to that address or the following addresses.

LGetPathname2 ($22)

Input:	UserID (2 bytes)	
	File Number (2 bytes)	
Output:	Address of Pathname (4 bytes)	
Errors:	$0000	- Operation succcessful
	$1101	- Entry not found
	$1103	- Pathname error
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function is identical to LGetPathname except that the output Pathname will be a Type 1 string rather than a Type 0 string.

UserShutDown ($12)

Input:	UserID (2 bytes)	
	Quit Flag (2 bytes)	
Output:	UserID (2 bytes)	
Errors:	$0000	- Operation succcessful
	$1101	- UserID not found
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function is called by the **Controlling Program** to close down an application which has just terminated. If the UserID specified is 0, the current UserID (USERID) is assumed.

The Quit Flag corresponds to the Quit Flag used in the GS/OS Quit call.

If the Quit Flag is 0, all Memory Blocks for the UserID are disposed and all the **System Loader**'s internal tables are purged of the UserID. The application can not be Restarted. The UserID is also removed from the system so that it can be reused.

if the Quit Flag is $8000, all Memory Blocks for the UserID are disposed and all the **System Loader**'s internal tables, except the Pathname Table, are purged of the UserID. The application can be reloaded but not Restarted because the pathname for the application is remembered.

If the Quit Flag is any other value, the memory blocks associated with the specified UserID (with Aux ID cleared) are processed as follows:

all memory blocks corresponding to Dynamic Load Segments are disposed
all memory blocks corresponding to Static Load Segments are made purgeable
all other memory blocks are purged

In addition, all dynamic segment entries in the **Memory Segment Table** and all entries in the **Jump Table List** for the specified UserID are removed. The application is now in a "zombie" state and can be resurrected by the **System Loader** very quickly because all the static segments are still in memory. However, as soon as any one static segment is purged by the **Memory Manager** for whatever reason, the **System Loader** must reload the application from its original Load File.

RenamePathname ($13)

Input:	Address of old pathname (4 bytes) Address of new pathname (4 bytes)	
Output:	none	
Errors:	$0000	- Operation succcessful
	$02xx	- Memory Manager error

This function will search the Pathname Table for a match on the old pathname specified and replace the matched pathname with the new pathname. The input pathnames must be Type 1 strings. The System Loader will call the GS/OS ExpandPath function for each input pathname before they are used for the string comparison and replacement. Therefore, the input pathnames can be full or partial pathnames of volumes, subdirectories or files.

GS/OS calls this function whenever it is told to change a pathname so that any files that the System Loader is managing have their pathnames changed also.

Jump Table Load

Input:	UserID (2 bytes)	
	Load File Number (2 bytes)	
	Load Segment Number (2 bytes)	
	Load Segment Offset (4 bytes)	
Output:	none	
Errors:	$0000	- Operation successful
	$1101	- Segment not found
	$1104	- File not Load File
	$00xx	- GS/OS error
	$02xx	- Memory Manager error

This function is called by an "unloaded" **Jump Table** entry to load a Dynamic Load Segment.

This function calls the Load Segment by Number function with the the Load File Number and Load Segment Number. If any errors occurred, the **System Loader** will report a System Death.

If the Load Segment by Number function has successfully loaded the segment, the **Jump Table** entry is made "loaded" by replacement of the JSL to the Jump Table Load function with a JML to the absolute address of the reference in the Dynamic Load Segment.

The **System Loader** will now transfer control to the absolute address.

Cleanup Routine

Input:	UserID
Output:	none
Errors:	none

This function is internal to the **System Loader.** Its function is to cleanup the **System Loader**'s internal tables in order to free memory.

If the UserID is 0, the **Memory Segment Table** is scanned for purged memory blocks and the handles to these blocks are disposed.

If the UserID is not 0, all Load Segments (both dynamic and static) for that UserID will be disposed. In addition, all entries for the UserID in the **Jump Table List** will be marked as unloaded.

Error Codes

$0000	Operation successful
$1101	Segment /Application/Entry not found
$1102	OMF Version error
$1103	Pathname error
$1104	File is not a Load File
$1105	not used
$1106	not used
$1107	File version error
$1108	UserID error
$1109	SegNum out of sequence
$110A	Illegal load record found
$110B	Segment is foreign

References

Apple IIGS ProDOS 16 Reference	--	Apple Computer
Apple IIGS GS/OS Reference	--	Apple Computer
Apple IIGS Programmer's Workshop	--	Apple Computer

GS/OS System Service Calls Delta ERS External

Version 0.01

By Bryan Atsatt

Revision History

Date	Version	Who	Description
1/31/89	0.01	BPA	Report name error for system service call $01FC54.

About This Document

This document describes changes and enhancements made since the 4.0 system disk release.

Cache_Flsh_Def ($1FC54)

The System Service Calls external ERS v0.11a01, page 4, incorrectly lists the call at $1FC54 as Cache_Lock. The call description on page 17 is correct.

Console Driver Delta

version 0.02

By
Rob Turner

Revision History:

version 0.01 03/02/88	first version.
version 0.02 03/14/88	changed Device Characteristics

This documents describes the two new calls that have been added to the Console driver as well as other changes to the console driver. The two calls that have been added are:

(1) Add_Trap
(2) Reset_Trap

In the area of performance, the Console driver is about eleven times faster doing single character writes. Most of the performance enhancements are due to work done on the device dispatcher and the GS/OS front end. Enhancements made in the Console driver include faster screen scrolling and faster write operations.

V INTERFACE CALLS

The GS/OS Console Driver supports the standard set of device driver calls. These are:

DRIVER_STARTUP
DRIVER_OPEN
DRIVER_READ
DRIVER_WRITE
DRIVER_CLOSE
DRIVER_STATUS
DRIVER_CONTROL
DRIVER_FLUSH
DRIVER_SHUTDOWN

It supports the standard status calls:

RETURN_DEVICE_STATUS
RETURN_CONTROL_PARAMETERS
RETURN_WAIT_STATUS

And the following driver–specific status calls:

GET_CONSOLE_INFO
GET_UIR_INFO
GET_TERMINATORS
GET_SCREEN_CHAR
GET_READ_MODE
GET_DEFAULT_STRING

The standard control calls are supported:

RESET_DEVICE
FORMAT_DEVICE
EJECT_MEDIA
SET_CONTROL_PARAMETERS
SET_WAIT_MODE

As are the following driver–specific calls:

SET_TERMINATORS
SAVE_TEXT_PORT
RESTORE_TEXT_PORT
SET_UIR_INFO
SET_READ_MODE
SET_DEFAULT_STRING
ABORT_INPUT
ADD_TRAP
RESET_TRAP

$0006 DRIVER CONTROL CALL

Input Parameters:	Device Number	≠ $0000
	Call Number	= $0006
	Control List Pointer	
	Request Count	
	Control Code	
Output Parameters:	Transfer Count	

This call is used to send control information to the console driver. The following control codes are legal:

$0000	Reset Device
$0001	Format Device
$0002	Eject Media
$0003	Set Control Parameters
$0004	Set Wait/No–Wait Mode
$8000	Set UIR Info
$8001	Set Terminators
$8002	Restore Text Port
$8003	Set Read Mode
$8004	Set Default String
$8005	Abort Current Input
$8006	**Add_Trap**
$8007	**Reset_Trap**

These calls are described more fully in a following section.

Possible Errors

$00	No_Error
$21	Drvr_Bad_Code
$22	Drvr_Bad_Parm
$23	Drvr_Not_Open
$32	**Add_Trap_Failed**

$8006 ADD TRAP

Takes the user supplied address and installs it in the console driver trap vector. The trap handler will be called by a JSL instruction. If the handler wishes to handle the call it should pull the return address off the stack, handle the call, then exit via an RTL instruction. When the trap handler is called the environment is set to the same environment as device drivers. furthermore, if the trap handler does not wish to handle the call it must restore ALL registers and direct page locations that it has used. To issue the call the user must set the request count to 4 and set the long word pointed to by the control list pointer to the address of the trap handler. If a trap is already installed an error will be returned.

The transfer count is 4.

Possible Errors

$32	Add_Trap_Failed

$8007 RESET TRAP

This call will remove a user installed trap vector if the vector is installed. The request count should be set to zero.

The transfer count is 0.

Possible Errors

$00	No_Error

The DIB contains the following fields. For more details please see the GS/OS Device Driver ERS.

NAME	DEFAULT	DESCRIPTION
Link Pointer	0L	Points to other DIBS which use this driver
Entry Vector	Start of code	Points to Console Driver's Entry Point
Device Characteristics	$0B60	Restartable,Loaded,CharDevice, RW enabled
Block Count	0L	n/a for character devices
Device Name	'.CONSOLE'	Default name of the console device
Slot #	3	Apple // 80 column drivers are in slot 3
Unit #	1	Only one device in slot 3
Device Version #	1	Driver version 1.0
Device ID #	$000A	Console is device 10

BASIC. System 1.3

Delta Document & Release Notes

Revision .01

****

Purpose -

This document is intended to provide an overview to the changes that have been made to the BASIC.SYSTEM file since version 1.2 was released. The areas of change for BASIC.SYSTEM include; 1) Version Level Change, 2) two bug fixes and 3) one minor enhancement. Each area will be addressed as separate items.

****

Version Level Change

The version number has been changed to reflect the newer version of the BASIC.SYSTEM file. Changes were made in the "Loader" source code to display the newer version number and in the Vernum segment in the "Tables" source code. Currently the version has been set to 1.3b, but the "B" will be removed when the code is final and fully tested.

****

Bug Fixes

Chain Command:

Problem- When two or more AppleSoft programs are chained together and the length of the variable table is a multiple of 256 (i.e. 256, 512, 768, 1024 ...) the chain command will not function correctly and a program error will occur in one of two ways. Either the program will crash into the monitor (if no onerr command has been issued) or the program will jump to an error routine if the onerr command has been used.

Solution- The solution for this was to patch the "MOVUP" command that is called by the Chain/Store function. Changed the value of A from 7 to 3 before chain is called and returned value to 7 after chain had been called. This alters a branch to include a decrement for the 'TO' and 'FROM' by $100 bytes before the move up routine is called.

Test Program - A sample test program has been written that allows the user to set the length of the variable table to verify the fixes implemented. When this program is run under BASIC 1.1 or 1.2 an error condition will occur whenever a multiple of 256 bytes is stored in the variable table. When this program is run under BASIC 1.3 the error will not occur no matter what the length of the variable table.

Changes Made - The changes were made in the following source files:
CI.BUFMGR --- added label C.SCMD
Tables --- added patch CHNCMD

****

****

Bug Fixes

BSAVE Command:

Problem- A problem with the BSAVE command occurs in version 1.1 and 1.2 of BASIC.SYSTEM where if you BSAVE over an existing file the old length and load address are retained. Note: This was not a problem in Version 1.0.

Solution- The solution to this problem is to combine the way Version 1.0 of BASIC.System worked with the way Version 1.1 & 1.2 work. This is accomplished by using the 1.0 approach whenever the "B" parameter is not used. When the "B" parameter is used, we instead go to the 1.1 & 1.2 implementation.

Test Program - To test this change, simply do the following under BASIC 1.0, 1.1, 1.2 and 1.3 to see the differences:

1) Create a sample Binary File at a desired memory location.
```
Create Myfile,TBIN
BSAVE Myfile,A$2000,L$400
```

2) Append File
```
BSAVE Myfile,A$4000,L$100, B$400
```

3) Replace beginning of file (B option used)
```
BSAVE Myfile,A$2000,L$80,B0
```

4) Revert to Original File size/data
```
BLOAD Myfile
BSAVE Myfile,L$400
```

Changes Made -

Changes were made in the following code segments:

CMDS1 - $AE4A Jump to First part of patch
$AEC9 Jump to Second part of patch
TABLES- $BB4C Patches start here and reproduce appropriate code

****

****

Minor Enhancements

Only one addition has been implemented in Version 1.3 of BASIC.SYSTEM, which is the addition of a command which allows a user to get into the Monitor without having to remember the specific call (CALL-151) number.

New Command : MTR or mtr [Direct Mode]

When issued from within BASIC, an MTR command will place the system at the monitor level. This command is identical to the Call-151 command that performs the same operation.

[Indirect Mode] : CHR$(4);"MTR"

When issued from within a BASIC program, the MTR command must be proceeded by a CHR$(4) (Control-D). Once issued the system enters the monitor and will stay there until a "Q", "q" or Control-C is issued by the user.

To return to BASIC from the monitor the user can either type; "Q" or "q", or Control-C.

Test Program - None, simply get into BASIC and type the commands.

Changes Made - Changes were made in the following code segments:

JSR to SRCHCMD1 added in CMDSYN1 segment. Jumps to code in Tables.

Code was added in the TABLES segment to test for and perform the monitor call.

****

GS/OS System Calls Delta ERS
External

Version 0.15

By Bryan Atsatt

Revision History

Date	Version	Who	Description
12/1/88	0.01	BPA	First descriptions of the Notification_Queue, Add_Notify_Proc call, Del_Notify_Proc call, Notify_Proc, and the @ prefix.
1/24/89	0.02	BPA	Description of new preference bits in set_sys_prefs and get_sys_prefs. Correction for os_shutdown description in ERS. Modified volume_change parameters.
1/26/89	0.03	RBM	Added system call D_RENAME.
2/10/89	0.04	BPA	Added system call Get_Std_Ref_Num.
2/28/89	0.05	BPA	Added system calls Get_Ref_Num and Get_Ref_Info, describe changes to Expand_Path, partial pathnames, bad_path_syntax errors, and correct error in original description of the Set_File_Info parameters.
3/2/89	0.06	BPA	Updated description of Get_Ref_Num, added info about valid file_sys_id's, added description of changes to Option_List parameter, added description of changes to the Create call.
3/3/89	0.07	BPA	Updated description of class 1 Set_File_Info call.
3/7/89	0.08	BPA	Added description of I/O redirection support and Quit call changes.
3/9/89	0.09	BPA	Added description of change to Format and Erase calls.
3/27/89	0.10	BPA	Added AppleShare ID to File System IDs list.
4/3/89	0.11	BPA	Added description of pcount parameter for Reset_Cache.
4/20/89	0.12	BPA	Updated status and control codes.
4/24/89	0.13	BPA	Updated Expand_Path description.
5/2/89	0.14	BPA	Added correction for D_Control.
5/11/89	0.15	BPA	Added correction for Get_Prefix.

About This Document

This document describes changes and enhancements made since the 4.0 system disk release.

The Notification_Queue

We have established a general notification mechanism for operating system events which allows applications to be notified when certain OS events occur. Two new Class 1 calls have been created to add and delete notification procedures to/from the Notification_Queue:

Add_Notify_Proc ($34)

This function adds a Notify_Proc (see description below) to the Notification_Queue. After successful installation, whenever the specified event(s) occur, GS/OS will call the notification procedure.

Offset	Label	Description
$00-01	pcount	input word value parameter count (must be 1)
$02-05	proc_pointer	input long word pointer pointer to the Notify_Proc to add to Notification_Queue

Errors: $04 invalid pcount
$53 parameter out of range

Del_Notify_Proc ($35)

This function deletes a Notify_Proc from the Notification_Queue. After successful deletion, the Notify_Proc will no longer be called.

Offset	Label	Description
$00-01	pcount	input word value parameter count (must be 1)
$02-05	proc_pointer	input long word pointer pointer to the Notify_Proc to delete from Notification_Queue

Errors: $04 invalid pcount
$53 parameter out of range

Notify_Proc

A Notify_Proc is composed of a header followed by code:

Name	Size	Description
Reserved	Long	Reserved (link to next task in queue).
Reserved	Word	Reserved.
Signature	Word	$A55A signature.
Event_flags	Long	Bit flags indicating which events to call procedure for:
Event_code	Long	The current event code will be put here by GS/OS. This will be equivalent to the Event_flags with the appropriate bit set (i.e. disk insert = $00000008).
Proc_Entry	????	Code. This will be JSL'd to in full native mode.

Bit	Event
0	reserved
1	switch GS/OS to ProDOS8
2	switch ProDOS8 to GS/OS
3	disk insert
4	disk eject
5	shutdown
6	volume_change (writing occured)
31-7	reserved

When Proc_Entry is called, it must set the direct page and data bank registers as needed; however, it does not need to save and restore the entry values.

In general, GS/OS calls cannot be made by the Notify_Proc (the OS is busy). The exceptions are the 'switch GS/OS to ProDOS8' and 'switch ProDOS8 to GS/OS' events; GS/OS calls are allowed during these events so that files may be opened or closed. Do NOT make a Quit call; the system will probably die a horrible death.

Notify_Procs may be called during an interrupt; therefore, keep the code short (set a flag or set up a Signal).

Parameters are passed to the Notify_Proc in the A, X, & Y registers. The parameters for each event are listed below:

switch GS/OS to ProDOS8	-	A = undefined. X = undefined Y = undefined
switch ProDOS8 to GS/OS	-	A = undefined. X = undefined Y = undefined
disk insert	-	A = device number X = undefined Y = undefined
disk eject	-	A = device number X = undefined Y = undefined
shutdown	-	A = undefined. X = undefined Y = undefined
volume_change	-	A = GS/OS call number X = device number Y = undefined The call number will be zero when the AppleShare FST detects that a server volume has been modified.

The @ Prefix

A new prefix has been added to help applications be 'AppleShare Aware'. This prefix is set by GS/OS whenever an application is launched. It is set to the folder where the application lives (same as prefix 9) unless the application is being launched from a server. In this case, the prefix is set to the user folder on the server.

This prefix should be used to access configuration files.

SET_SYS_PREFS ($0C)

Two new preference bits (14 & 13) have been defined for controlling OS dialogs.

Offset	Label	Description
$00-01	pcount	input word value parameter count (must be 1)
$02-03	preferences	input word value of system preferences: bit 15=0 do not display volume mount dialog bit 15=1 display volume mount dialog bit 14=0 use standard volume mount dialog bit 14=1 use volume mount dialog without cancel button bit 13=0 do not suppress error dialogs (those with only 1 button, such as the "disk damaged" dialog). bit 13=1 suppress error dialogs. bits12:0 reserved, must be zero.

Errors: $53 parameter out of range

GET_SYS_PREFS ($0F)

Two new preference bits (14 & 13) have been defined for controlling OS dialogs.

Offset	Label	Description
$00-01	pcount	input word value parameter count (must be 1)
$02-03	preferences	output word value of system preferences: bit 15=0 do not display volume mount dialog bit 15=1 display volume mount dialog bit 14=0 use standard volume mount dialog bit 14=1 use volume mount dialog without cancel button bit 13=0 do not suppress error dialogs (those with only 1 button, such as the "disk damaged" dialog). bit 13=1 suppress error dialogs. bits12:0 reserved, must be zero.

Errors: none

GS/OS SCSI Driver (General) External ERS

Ver. 5.0 b03

** PRELIMINARY**

Written by: Matt Gulick

Revision History

Date	Version	Description of Revision
03-08-1988	0.00a00	Document Started
03-10-1988	0.00a00	Added details of SCSI Driver startup and tables internal to the driver for efficient running.
03-24-1988	0.00a00	Added detailed description of Startup call and how to get the information needed to build the DIBs.
03-28-1988	0.00a00	Updated previous information and continued to explain the calls to the driver.
03-30-1988	0.00a00	Fleshed out Driver Descriptions.
04-11-1988	0.00a00	Update and continue Driver Definition.
04-14-1988	0.00a00	Started describing the device specific status codes. First attempt at describing how linked commands are managed.
04-18-1988	0.00a00	Continue Status Call descriptions.
04-19-1988	0.00a00	Continue Status Call descriptions.
04-20-1988	0.01a00	Completed Status Call descriptions. Document released and Control Call descriptions started.
04-21-1988	0.02a00	Fleshed out the three standard calls dealing with partitions. Started to define the device specific control codes.
04-25-1988	0.03a00	Continuing the control calls starting with $8011 Space command.
04-27-1988	0.04a00	Continuing the control calls starting with $802A Read command.
05-04-1988	0.05a00	Document finished and circulated for general review.
05-05-1988	0.05a00	Modified all Request counts that specified a block count. The SCSI command requires a block count, but the application will specify a byte count. It is up to the driver to make the conversion. The affected fields show $00xxxxxx as their maximum length as opposed to the more standard hex representation.
05-16-1988	0.06a00	Marked some commands as unsupported. This is due to the conflict they implicitly have with the GS/OS environs.
08-17-1988	0.08d01	Review and revised to reflect the current state of affairs.
09-11-1988	0.09d01	Review and revise. Add section detailing the usage of data chaining commands.
09-28-1988	0.10d1	Revise information, clarify what standard calls are supported and release as an External ERS.

01-10-1989	0.11d1	Fix documentation errors, typos, and update to reflect Revision 6 of the SCSI-2 Standard.
02-15-1989	0.12d1	Fix documentation errors, typos.
03-07-1989	0.13d1	Fix documentation errors, typos and make some sections clearer to the reader. Added notes to the section on Device Specific Commands and added the RESERVE UNIT Command for the Apple Tape Drive.
04-19-1989	5.0 b02	Cleaned up some info dealing with the Data Chaining Commands. Also modified description of error conditions in both the READ and WRITE Calls. Moved $1B 'SCAN', $2E 'WRITE AND VERIFY', and $2F 'VERIFY' commands from Device Specific Status Calls to Device Specific Control Calls.
04-24-1989	5.0 b03	Problem with READ CAPACITY $8025 fixed.

About This Document

This document explores the task of writing an SCSI (Small Computer Systems Interface) Driver for GS/OS following the block diagram in **Figure 1**. It is assumed that the reader is somewhat familiar with the ANSI® x3.131-1986 and SCSI-2 document and SCSI in general. It is also recommended that the reader have available the most recent revision of the SCSI-2 draft or final spec. This document is necessary because of the devices that have been added to SCSI since the release of the SCSI-1 Spec.

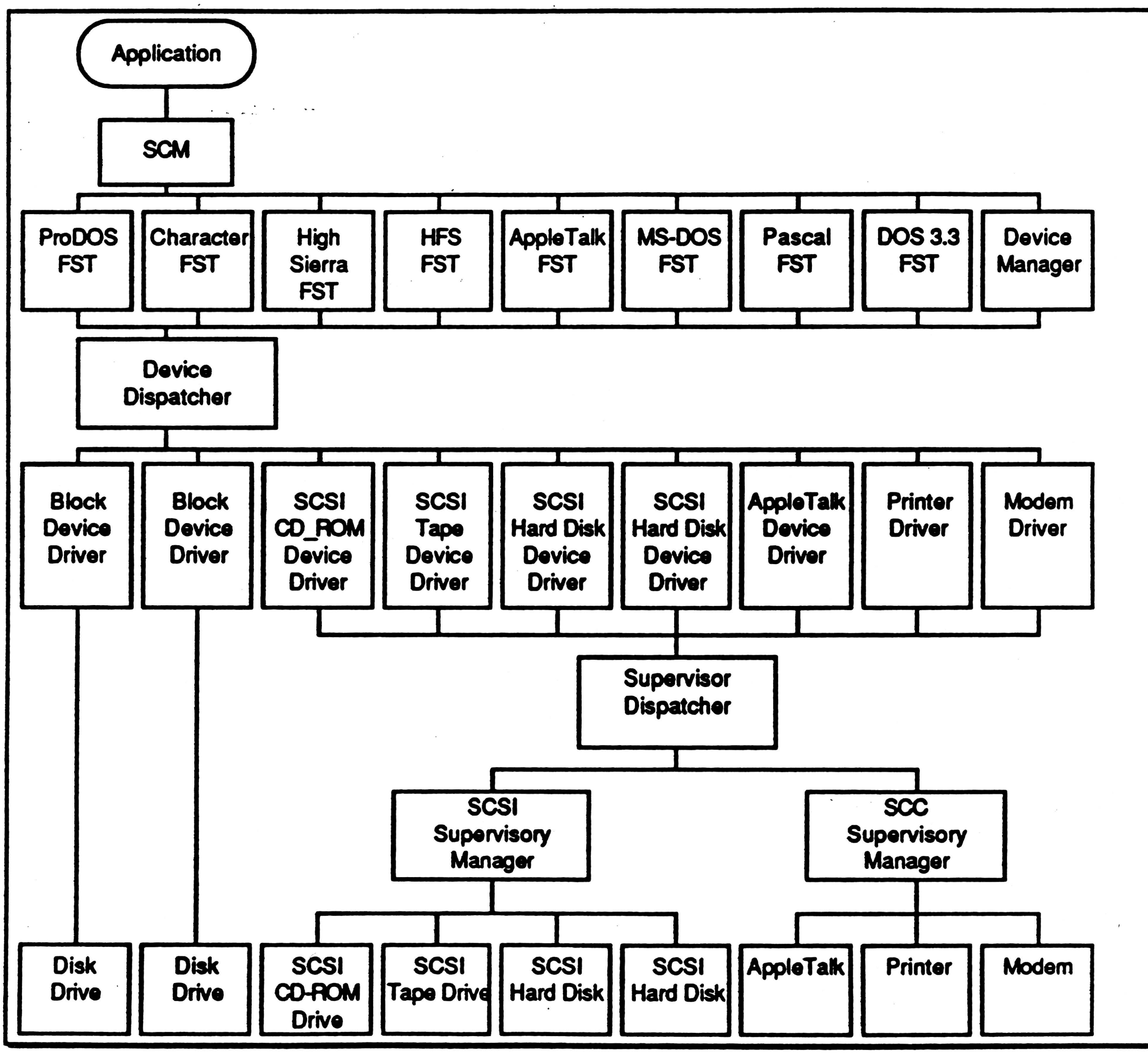


Figure 1

Physical vs. Logical Devices

In the world of SCSI, the actual box containing the hardware and connecters is considered to be an SCSI Device. Each SCSI Device can consist of one or more Logical Units. An example would be a 40 Mb fixed hard disk with a built in 5 Mb removable hard disk cartridge for backup purposes. These Logical Units can consist of Block and Character Devices in any combination. Because of the way the Finder will represent these devices, they are treated as separate and individual *Physical Devices* in this document. Block Devices can further be partitioned into separate volumes within that device. These are called *Logical Devices* in this document.

Loading the SCSI Driver

The SCSI Driver is a loaded device driver and as such must adhere to a few rules.

As stated in the ***GS/OS Device Driver ERS*** the SCSI Driver is compacted to object module format type 2.

It has a file type of $BB.

In addition, the AuxType for the SCSI Driver is $000001xx. This indicates that the driver is active, that it is indeed a driver, and that at load time there are xx devices supported. The 'xx' represents a value in the range of $01 - $3F. This is the number of devices that will be started up at the time that the driver is loaded. This number depends on the group of devices targeted by the driver being loaded. If the device is a character device, this number will be small ($01, $02 or greater). If the targeted devices are block devices, there is a possibility that $3F devices can exist on the system. A worst case example is 8 slots * 7 SCSI Devices * 8 Logical Units * 63 partitions on each of them. This result is $6E40 Logical Devices. Unrealistic not to mention insane but still possible. At load time the driver can build DIBs for as many of the devices that are online as it wants, but only the first $3F will be started by the Device Dispatcher. It is then up to the driver to ensure that any remaining devices are also started at a later time using the Post Driver Install call. A method for this is discussed in the Driver Startup Call discussion.

Physical Structure of an SCSI Driver

An SCSI driver consists of a driver header, configuration script, configuration parameter list, Device Information Block(s) (DIB) and the driver code segment. If the device driver supports more than one device type, a configuration script, configuration parameter list and DIB must be provided for each device. Each device may have it's own individual code segment or a common code segment may be shared by all devices supported by the driver.

About the Driver Header

The header is used when loading the driver. It indicates where the configuration parameter lists and DIBs are located. The device dispatcher loads only the driver, DIBs and configuration parameter lists using an initial load call to the system loader. The header contains the following information:

Word	Offset to 1st DIB
Word	Count of number of devices
Word	Offset to 1st configuration parameter list for device #1
Word	Offset to 1st configuration parameter list for device #2
Word	Offset to 1st configuration parameter list for device #3
Word	Offset to 1st configuration parameter list for device #4
etc.	

About the Configuration Scripts

The structure of the configuration scripts is not completely defined at this time. The current drivers define 2 long words of zero as the Configuration Parameter Lists.

Please refer to the ***GS/OS Device Driver ERS*** for a discussion of this information.

About the Configuration Parameter List

The Configuration Parameter List contains device dependent information used in configuring a specific device. The configuration parameter list may be pre-configured through the use of the configuration program. Additionally, the configuration parameter list may be examined or modified by the status and control calls supported by the device driver.

In addition to the configuration parameter list a default configuration parameter list should be located contiguous to each configuration parameter list.

Please refer to the ***GS/OS Device Driver ERS*** for a discussion of this information.

About the DIB

The DIB must contain the following information:

Link Pointer.
Entry Pointer.
Device Characteristics.
Block Count. *It should be noted that this parameter may be dynamic if the device supports assorted types of removable media or partitioned removable media. In this case any status, or*

	control call that detects online and disk switched should update the block count in the DIB after media insertion.
Device Name.	*The device name must be in upper case with the MSB off.*
Slot Number.	*Bits 0 through 2 indicate the slot while bit 3 indicates that the slot is internal or external.*
Unit Number.	
Device Version Number.	
Device Type ID.	*A list of device types and their associated ID numbers is shown below:*

$0000	Disk][(includes Duodisk, Unidisk & Disk//c)
$0001	Profile 5 meg
$0002	Profile 10 meg
$0003	*Disk 3.5*
$0004	SCSI (generic)
$0005	SCSI HD
$0006	SCSI Tape
$0007	SCSI CD-ROM
$0008	SCSI Printer
$0009	Modem
$000A	Console
$000B	Printer
$000C	Serial LaserWriter
$000D	AppleTalk LaserWriter
$000E	Ram Disk
$000F	Rom Disk
$0010	File Server
$0011	IBX
$0012	AppleDesktop Bus
$0013	Hard Disk - (generic)
$0014	Floppy Disk - (generic)
$0015	Tape Drive - (generic)
$0016	Character - (generic)
$0017	MFM Floppy Disk - Super Drive
$0018	Network (generic - AppleTalk)
$0019	Sequential access device
$001A	SCSI Scanner
$001B	Other Scanner
$001C	LaserWriter SC
$001D	AppleTalk Main Driver
$001E	AppleTalk File Service Driver
$001F	AppleTalk RPM Driver
$xxxx	Apple 40 MB Tape Drive
$xxxx	SCSIoid this an that Driver

Note: *The list of device types provided above does not in any way indicate that Apple Computer, Inc. intends to provide devices or device drivers for the types listed. In creating this list, we tried to think of as many unique device types as possible that may need a unique device type ID*

assignment. Device type ID numbers are assigned by Apple Computer, Inc. If you are developing a driver for a device type that is not listed, you will need to get a device type ID assignment from Apple Computer, Inc.

Head Device Link.
Forward Device Link.

Note: *Both link fields are maintained on a by device basis. These links imply a link between two or more logical devices within a single physical device. A value of zero indicates no link.*

DIB Extension Ptr. *This long word pointer points to where the Extended data of the DIB Begins.*

DIB Device # *This is the device number assigned to this DIB by the OS and should never change, even when the device goes offline, until the DIB receives a valid shutdown call.*

These fields are discussed in detail as it applies to an SCSI Driver in the section of this document dealing with the actual building of the DIBs.

Logical Structure of an SCSI Driver

The SCSI Driver has a main entry point (**Item 'a', Figure 2**) that when called decides the type of call being issued and routes the call to the appropriate call filter (**Item 'b', Figure 2**). These filters, depending on the call, gather information from one location and route it to another. This could be information from the Application to the device or vice-verse. The call may also result in the DIB being updated with any changes at that time.

If the filter needs to talk to the device, it issues calls to the SCSI Device sending them via the Main Driver (**Item 'c', Figure 2**) section of the overall device driver. Each call to the main driver results in one or more calls to the Supervisory Dispatcher (**Item 'e', Figure 2**) for routing to the SCSI Supervisory Manager (SCSI Manager) which uses this data to communicate with the target device via the hardware. The drivers that are written correctly will never have carnal knowledge of the hardware environment, and if any hardware changes are made, the drivers will be completely unaffected.

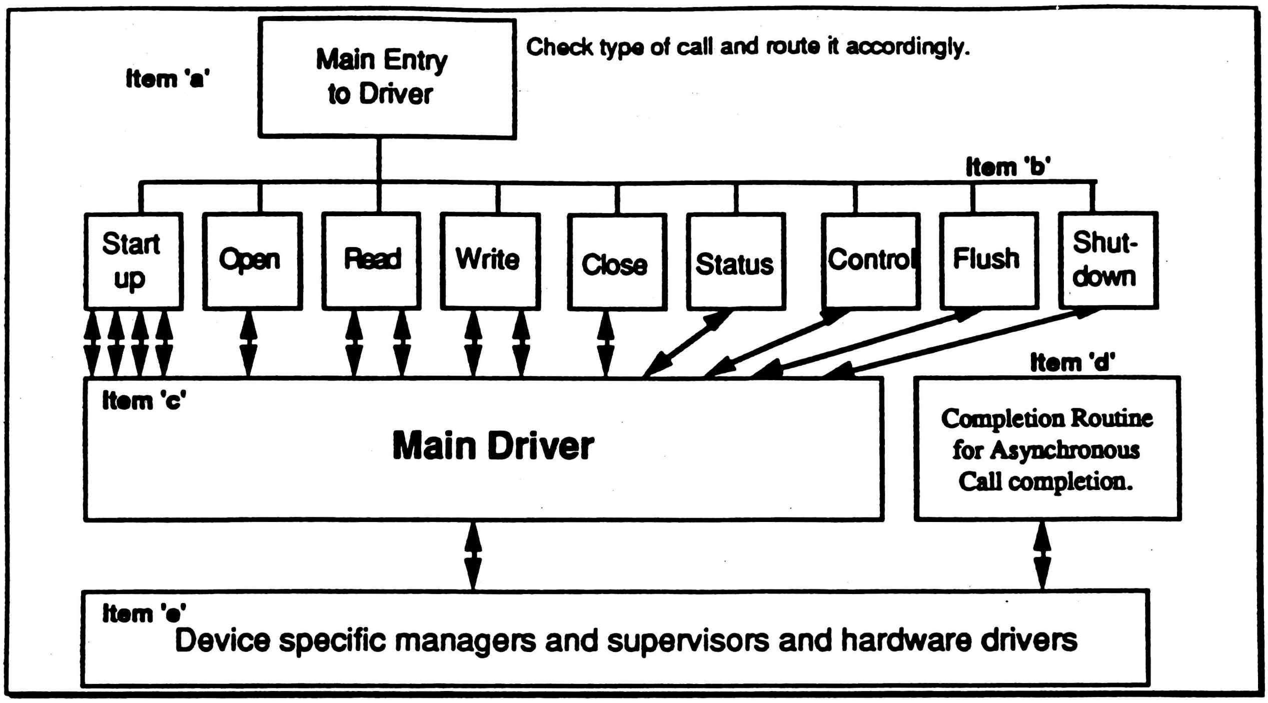


Figure 2
(Internal structure of SCSI Driver)

General Function of the Driver

The SCSI Driver, like all other drivers, translates calls from an FST via the Device Dispatcher into a format understood by the SCSI Manager. This is the same set of calls that are sent to other drivers and their input/output data structure are as outlined in the picture 'CALLS TO DEVICES'.

Driver Calls

All SCSI Drivers accept a standard set of calls. These calls include:

DRIVER_STARTUP	*(Not an Application Call)*
DRIVER_OPEN	
DRIVER_READ	
DRIVER_WRITE	
DRIVER_CLOSE	
DRIVER_STATUS	
DRIVER_CONTROL	
DRIVER_FLUSH	*(Not an Application Call)*
DRIVER_SHUTDOWN	*(Not an Application Call)*

The details of each driver call will be described individually, and are described in detail on the following pages.

CALLS TO DEVICES

Offset	DRVR_STARTUP	DRVR_OPEN	DRVR_READ	DRVR_WRITE	DRVR_CLOSE	STATUS	CONTROL	FLUSH	SHUTDOWN
$00–$01	DEVICE NUMBER	DEVICE NUMBER	DEVICE NUMBER	DEVICE NUMBER	DEVICE NUMBER	DEVICE NUMBER	DEVICE NUMBER	DEVICE NUMBER	DEVICE NUMBER
$02–$03	CALL NUMBER	CALL NUMBER	CALL NUMBER	CALL NUMBER	CALL NUMBER	CALL NUMBER	CALL NUMBER	CALL NUMBER	CALL NUMBER
$04–$07			BUFFER POINTER	BUFFER POINTER		BUFFER POINTER	BUFFER POINTER		
$08–$0B			REQUEST COUNT	REQUEST COUNT		REQUEST COUNT	REQUEST COUNT		
$0C–$0F	TRANSFER COUNT	TRANSFER COUNT	TRANSFER COUNT	TRANSFER COUNT	TRANSFER COUNT	TRANSFER COUNT	TRANSFER COUNT	TRANSFER COUNT	TRANSFER COUNT
$10–$13			BLOCK NUMBER	BLOCK NUMBER					
$14–$15			BLOCK SIZE	BLOCK SIZE					
$16–$17			FST NUMBER	FST NUMBER		STATUS CODE	CONTROL CODE		
$18–$19			VOLUME ID	VOLUME ID					
$1A–$1B			CACHE PRIORITY	CACHE PRIORITY					
$1C–$1F			CACHE POINTER	CACHE POINTER					
$20–$23	DIB POINTER	DIB POINTER	DIB POINTER	DIB POINTER	DIB POINTER	DIB POINTER	DIB POINTER	DIB POINTER	DIB POINTER

In a few cases some of these calls result in nothing more than returning to the caller. An example is the DRIVER_OPEN call to a block device driver. Other calls cause the driver to set internal flags for future reference (ie. altering wait/no-wait mode). Yet other calls result in a single or even several calls being sent on down the line to the target device.

At the completion of the call the DIB may be updated with new information, the requested data are returned from the device if a GET_INFO, READ or a STATUS call was issued or data are sent on to the device in the case of an SCSI FLUSH, WRITE or a CONTROL call.

It is also up to the driver, in the case of an SCSI Block Device, to be aware of and account for the partition map of a device. It is the responsibility of SCSI Block Device drivers to interpret the information in the partition map if it exists. This is done not only at startup time, but also if a disk switched condition exists or an application issues a Write_PMap, or Assign Partition calls to the device in question. The later three situations cause the driver to re-build the DIBs belonging to that device. This is discussed in detail later in this document.

The device specific Status and Control calls available to the File System Translators (FSTs) as well as applications have a structure after which all calls to the Main Driver are structured. Even the Startup, Read, and Write calls along with the other types of calls are modified into a single or group of SCSI Status and/or Control calls. These calls use the same control and status codes as defined for the outside world while talking to the driver.

There are of course codes defined that are at first glance in violation of the spirit of the status and control calls. This was necessary for device intelligent applications that need to issue some of the more powerful SCSI supported calls. The read call is translated to a status call and the write to a control call. To allow for this the Command Codes are styled after those defined by the ANSI® X3.131-1986 and SCSI-2 documents. The text below was extracted from the former.

The operation code (Table 6-1) of the command descriptor block has a group field and a command code field. The three-bit group code field provides for eight groups of command codes. The five-bit command code field provides for thirty-two command codes in each group. Thus, a total of 256 possible operation codes exist. Operation codes are defined in sections 7 through 13.

Group 0 - six-byte commands (see Table 6-2)
Group 1 - ten-byte commands (see Table 6-3)
Group 2 - reserved
Group 3 - reserved
Group 4 - reserved
Group 5 - twelve-byte commands (see Table 6-4)
Group 6 - vendor unique
Group 7 - vendor unique

The Driver uses the least significant byte of the control and status codes to be the SCSI command and in the most significant byte we use the high bit to indicate a device specific call. All other bits are currently reserved by Apple Computer, Inc. Please see **Figure 3** below for a bit by bit example of the Command Code structure.

It is important that the Application MUST verify the Device ID before issuing any device specific call.

Because the command codes are structured this way, there are holes in the command code list for the Control Calls as well as the Status Calls. It is entirely possible for the two sets of calls to be put together in a single list without any duplication of command numbers. Because of this, the driver must make the distinction of what qualifies as a Control Call and whether to allow that call to be also issued as a Status Call. This is also true in the reverse situation. It is possible for a command number to be a Status Call to one type of device and a Control Call to another type. Drivers that handle multiple device types will need to distinguish one from the other.

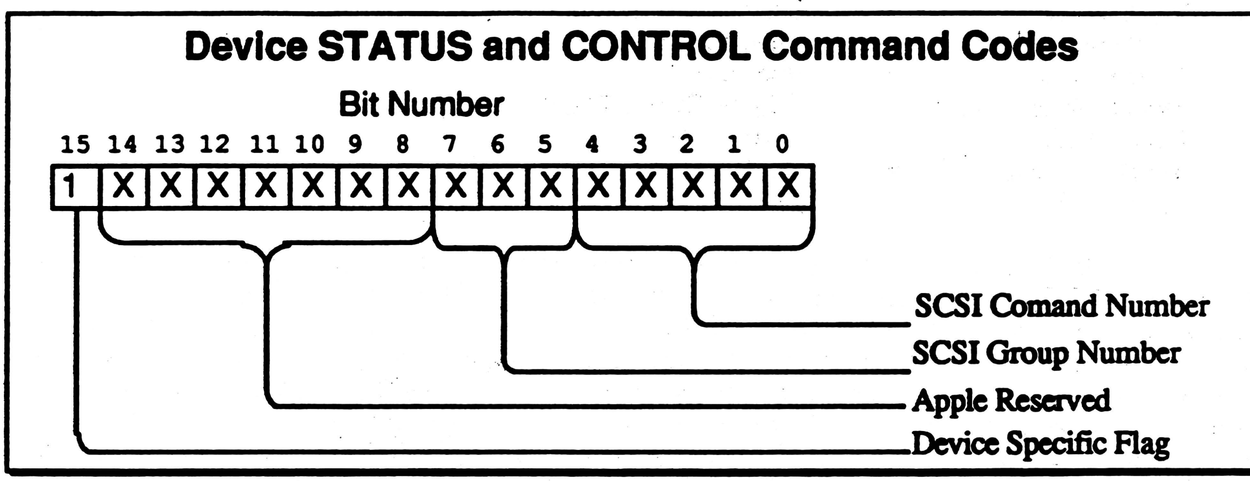


Figure 3

Caching Data Blocks

!!!!!This section is still under review. When the design is worked out it will be added to this document here!!!!!

Sparing

Sparing is used to reassigning a block that has a lowered read reliability, all failed writes will be spared, and reads are spared only when the data can be retrieved. The following algorithm should clarify what actually takes place.

A read call is issued to the device specifying a particular block. The device reads the block checking the data for correctness. If the data is damaged, an error is returned, and the call retried. If this retry is needed more and more before the correct data is

returned, then the block is reassigned and the data is written to the new block before returning. If the block is so bad that the data could not be read, then an error is returned to the caller and the block is not spared. (It is possible that a later read will be successful). If the block is damaged, then the next write will cause the block to be spared. This is due to the inherently destructive nature that a write has over the previous data.

How Device Drivers Get Called

The SCSI Device Driver's main entry point is called by the Device Dispatcher in full native mode (16 bit 'm' & 'x') via a 'JSL' instruction. Prior to calling the SCSI device driver, the device dispatcher sets the transfer count to zero. The processor state prior to device driver dispatching is set as follows:

Accumulator	=	Call Number
Y Register	=	Unspecified
X Register	=	Unspecified
P Register	=	0=m=x=e
Direct Page	=	GS/OS Direct Page
Data Bank	=	Unspecified
Stack Pointer	=	GS/OS stack
System Speed	=	Fast Mode

The data bank is preserved by the Device Dispatcher. A standard parameter block is set up on GS/OS direct page prior to dispatching to the SCSI device driver. Currently, there is no workspace available for a drivers use on GS/OS direct page. Any SCSI Drivers requiring direct page must roll their own either when the driver is started or as a result of an open call. The driver must release this direct page as a result of either a shutdown or close call respective to which call acquired the direct page. Device drivers must not permanently modify any GS/OS direct page location with the exception of the transfer count which indicates the number of bytes processed by the driver.

DEVICE DRIVERS MUST NEVER ACCESS GS/OS DIRECT PAGE USING ABSOLUTE OR ABSOLUTE LONG ADDRESSING MODES (Rumor has it we will move the location of GS/OS direct page with every release!!!). To access the GS/OS Direct page the driver saves the direct Page address and retrieves it for use as an index when needed (See code example below).

```
tdc                     ;Get Direct Page address
sta   |scsi_dp
      .
      .
      .
ldx   |scsi_dp
lda   >call_number,x
```

SCSI Drivers must return via an 'RTL' instruction in full native mode (16 bit 'm' & 'x'). The register states on return to the device dispatcher are as follows:

Processor Status	=	Carry set if error, Carry cleared if no error
Accumulator	=	Error code if error, $0000 if no error
Y Register	=	Unspecified
X Register	=	Unspecified
Direct Page	=	GSOS Direct Page
Data Bank	=	Unspecified
Stack Pointer	=	GSOS stack

Driver Startup Call

Call Parameters :	Device Number	≠	$0000
	Call Number	=	$0000
	DIB Pointer		

Device Number. *This word parameter specifies the target device. This parameter must be nonzero.*

Call Number. *This word parameter specifies the type of call. Must be a zero for this call.*

DIB Pointer. *This longword points to the DIB for the device being accessed. For the first time that call is issued, this points to the DIB structure included with the driver when it was loaded.*

This call is supported by both character and block SCSI drivers and is executed by GSOS during initialization or after loading a driver in preparation for launching an application. This call performs any tasks necessary to prepare the driver for operation. This includes memory allocation. Character device drivers maintain an internal flag indicating when the device is open; therefore, the open flag should be set to <u>not open</u> by this call.

<u>**This call must not be issued either by an Application or an FST!!!**</u>

The device dispatcher sets the DIB pointer on direct page and in the devices DIB it sets the DIB_DEVICE_NUMBER prior to issuing a startup call to the device. This parameter is used by devices that support removable partitioned media. Each media partition will be accessed through a separate device driver[2] as if that partition was a separate device. Since multiple devices can share a common media it is necessary to maintain links between these devices in order to manage disk switch and off line conditions that may have to be reflected at each linked device. **!!!!!The device driver is responsible for maintaining these device links and uses the DIB device number to initialize the head link and forward link in the DIB!!!!!** It should be noted that if the driver is replacing the boot device, the DIB device number

---

2 Each DIB is considered to have it's own separate driver, but it may actually be that one common driver code segment services more than one DIB.

will not remain constant. The slot and unit number contained in a DIB is not considered valid by the device dispatcher until the device has successfully returned from the startup call. If the slot and unit in the DIB match the boot device's slot and unit then the driver will be relocated in the device list to device #1. In this case, the DIB device number will be updated to device #1 but only after startup has been completed. In this case the driver can only consider the DIB device number to be correct on the second device access.

Details of the Call.

When GS/OS is started, it finds the SCSI Driver in the DRIVERS sub-directory, loads it and issues a DRIVER_STARTUP call. The driver startup call does several things at this time to prepare the driver for use in the GS/OS environment. As part of the driver object code is a template for the building of the DIBs needed for the devices that the driver will be in communication with. In **Figure 4** is a layout of the DIB as described in the Device Driver ERS.

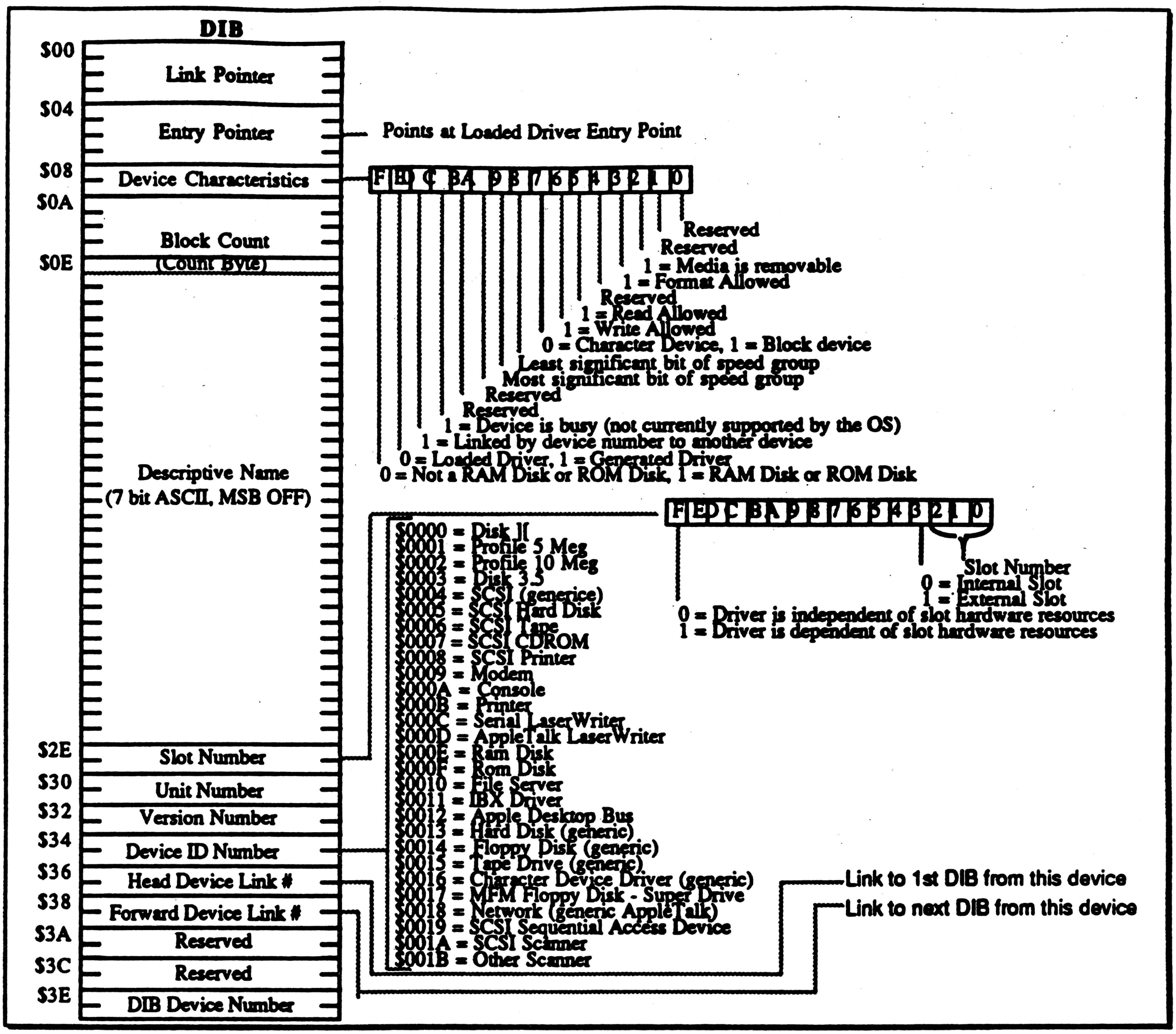


Figure 4

Some of the information in the DIB is standard for all DIBs associated with this driver and are setup as data statements in the DIB template used by the driver. This template could also double as the first DIB if the driver is unsuccessful in finding devices to communicate with. By so doing, it is possible for the driver to remain active rather than being purged if there are no devices active. This way if a device becomes active and we are notified, the driver will establish communication with the device without rebooting.

In addition to the standard fields defined for a DIB are several other fields that aid in the management of the SCSI Device for which the driver was written. These extensions are driver dependent, and just because one driver uses them, not all need to. For the GS/OS Drivers under development at Apple for SCSI Devices these structures will be used as defined. The DIB Extension as defined for this driver are discussed in detail in the section titled 'DIB EXTENSION DATA'.

What the driver does.

On startup, the driver will need to get the ID for the SCSI Manager then issue a Request_Devices call to the SCSI Manager through the Supervisor Dispatcher. The parameter list (**Figure 5**) will be used by the SCSI Manager to build a list that is returned in the buffer specified.

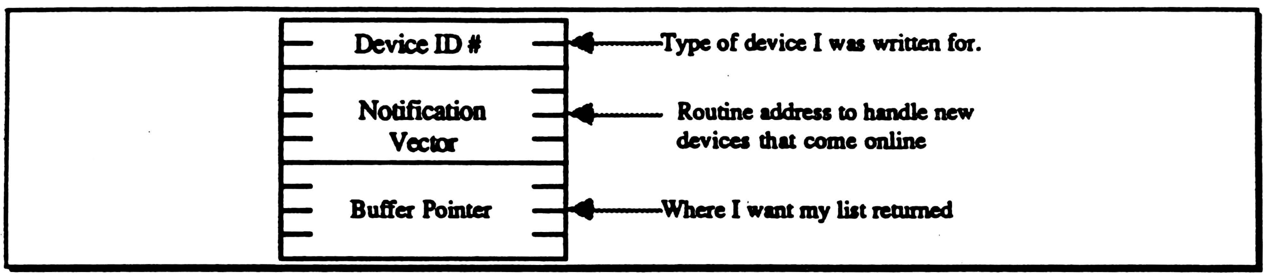


Figure 5

The *Device ID #* is of the same set of IDs as those defined by the SCSI inquiry call. This is to tell the SCSI Manager what types of devices we want to identify. The SCSI Manager will return to the driver a list of the currently connected devices of this type. It is from this list that we well build our DIBs.

The next field is the Event Notification Vector. This is the address of the routine that the SCSI Manager should call when a new device comes on line that is of the type specified in the device type field, or any other async event notification.

The last field is the buffer to which the list of devices would be returned.

The return data (**Figure 6**) starts with a word pointer. This is the area that has been set aside for use by the driver as its direct page. It is 256 bytes long and is shared by all SCSI drivers. All data stored here by the driver should be considered to be destroyed upon exiting the driver. The next word is the device count. This is the number of entries contained in the buffer. The list itself is made up of a two word pair for each device returned . The first word is meaningless and is used to fill in the 'SLOT' word in the DIB. The second word is the unit number. This is the value we will use when calling the SCSI Manager. The low 10 bits are reserved for use by the SCSI Manager. The high 6 bits can be used by the driver but must be zero when calling the SCSI Manager. These are used by the driver of block devices to give partitions a unique unit number. If this were not done, the slot and unit number of all the partitions on a device would be the same and all but one of them would be shut down.

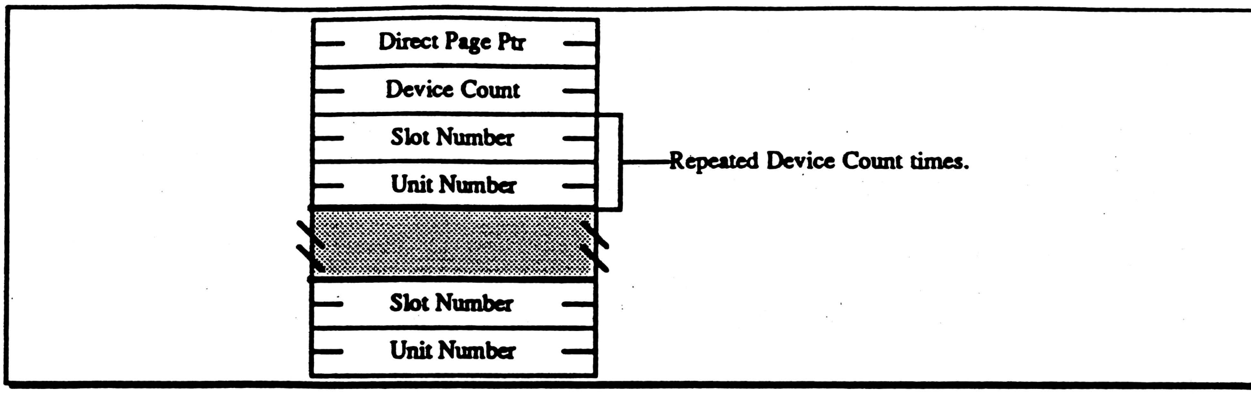


Figure 6

The driver now knows how many devices are currently attached to the system. It must be noted that these devices are physical devices as far as the driver is concerned. Even though only 1 device is returned, it may be that there are 10 partitions on it. This will only be possible on block devices and the CD-ROM. The Operating System itself will not be able to distinguish between a physical or a logical device except by checking the linked device bit in the device characteristics.

As each DIB is built it will be necessary for the driver to issue a few SCSI calls to the device to get the needed information. The method of discovery for this information will be discussed in two ways. First will be a field by field description of what is needed and what calls to issue to get the information. Second will be a discussion of the SCSI calls needed and what data requirements they will fulfill. This will hopefully reduce any confusion as to how the information for the DIB is found. Also all pictures will be given in the section detailing the SCSI calls. They will have more meaning there.

Building the DIBs field by field.

Link Pointer: This field is used to connect the DIBs for this driver to each other for management purposes. This will point to the next DIB built in sequence and should never change unless the memory where the DIB is located is freed up. This would be done through the memory manager and is not recommended. If a DIB is no longer in use, it should then be marked first as a disk switch then as off line. This Memory is now available to the driver if a DIB space is needed for this physical device only.

Entry Pointer: This is nothing more than a pointer to the drivers main entry point.

Device Characteristics: The default for this field in binary would be $00x000110xx0xx00 for a character device and $00x000111xx0xx00 for a block device where x represents information we do not yet know.

The first x (left to right) is the linked DIB bit. This is used to indicate that this device contained two or more partitions. When one of the DIBs is ejected, and this bit is set, the the DIB will be marked offline and those linked DIBs will remain online. Ejection will only take place when the last online DIB is ejected.

The next x is *Write Allowed.* This is returned by the MODE SENSE command. Bit 7 of byte 2 in the 4 byte header set to 0 indicates that the media is write allowed. See the ANSI® X3.131-1986 SCSI document on page 110.

The next x represents the *Read Allowed* bit. **!!!!! As far as I can tell at this time, all SCSI devices are read allowed except maybe an SCSI Printer but then the driver will know who he is talking to. Right? !!!!!**

Next in the lineup is the *Format Allowed* bit. This bit is not as easy as the previous bits but is still readily available to the driver. This is found by first issuing an INQUIRY call then by checking the group 0 call bitmap for bit 3 of byte 0. If this bit is set then the device will accept FORMAT calls.

NOTE: Refer to command bitmap discussion.

Lastly, is the media removable? This is contained in the data returned by the INQUIRY call also. Bit 7 of byte 1 in the 5 byte header if set to 1 indicates that the media is removable.

Block Count: This information is taken from the READ CAPACITY Call in the case of block devices. If the block device contained several partitions, then this information will be extracted from the partition map entry for this logical device.

Descriptive Name: This field contains the device name that is used by the Operating System. In the case of a hard disk this will be 'APPLESCSI.HD00.00' where the first two 0's following the HD refer to the device number and the two 0's following refer to partitions on that device. For other devices, the names are similar and indicate the type of device. Refer to the following list:

APPLESCSI.HD00.00	Apple Hard Disk and other direct access devices.
SCSI.TAPE00.00	Sequential Access Devices
APPLESCSI.Printer00.00	SCSI Printer Devices
APPLESCSI.Proc00.00	SCSI Processor Devices (ie. other computers, ect)
APPLESCSI.WORM00.00	Write-Once Read-Many
APPLESCSI.CDROM00.00	CD-ROM Devices
APPLESCSI.Scanner00.00	SCSI Scanners
APPLESCSI.Optical00.00	Optical Memory
APPLESCSI.Changer00.00	Changer Devices (ie. jukebox, ect)
APPLESCSI.Com00.00	Communication Devices
APPLESCSI.TAPE00.00	Apple's Tape Drive & 3M MCD-40/SC Tape Drive

Slot Number. This is returned as part of the data from the Get_Devices call to the SCSI Manager. It has no meaning to the SCSI Driver.

Unit Number. This is also returned by the Get_Devices call but it does have meaning for us. This two byte value is in reality two fields put together to form the

unit number. The least significant 10 bits are the physical device number as it appears to the SCSI Manager. This has no meaning for us except in the case of a device going off-line or if the media is removed. We need some way of identifying all the devices that we have built DIBs for that are affected by this occurrence. This will be discussed later in full detail. The most significant 6 bits represents the logical device or partition on the actual physical device.

Version Number: This, as discussed above, is the version of the driver that we have written.

Device Number: Also discussed previously, this is the number associated with the type of device our driver was written for.

Head Link Device Number: This is the device number used as an entry to the device list where the pointer to the first DIB for this device resides in memory. Each device will have a series of DIBs depending on the number of partitions residing on that disk.

Forward Link Device Number: This is the device number used to locate the next DIB that follows this one in our set of DIBs for the Physical device.

DIB Extension Pointer: This is a long word pointer that points to an area used for additional data about the device that the driver needs to maintain for it's operation.

DIB Device Number: This is the number used to reference this DIB and is used in the Forward and Head Link Device Number fields.

DIB Extension Data.

In each DIB built by this driver, there is an extension that contains data required by this driver to function in the best possible manner. Each field used by this driver is discussed below.

The order in which these are presented is not necessarily the order of their occurrence. There are at least two reasons for this.

First, things change. This is not a problem as long as the code knows what is taking place. And since the code built the extension, this is not a problem.

Second and most important. The Driver is the only thing in existence that should be looking at what is stored there in the first place.

Starting Block Number: This long-word value is used only when partitions are present. Normally, 0's will be stored here. When a request is made to read a specific block number, this value will be added to give us the physical block number on the device where the data requested resides. For character devices, this has no meaning.

Head Pointer: This three byte field is an actual address for where the first DIB for this device can be found. This again is only used when partitions are present.

Forward Pointer: This three byte field points to the next DIB associated with this device in the same manner as the Head Pointer points to the first DIB for this device.

Memory DIB Count: This word value is only valid when within the first DIB of this memory segment. It is used to indicate how many active DIBs remain. During a shutdown call to any DIB within this memory segment, this value will be decremented and upon reaching zero, this memory segment will be released via the memory manager.

Device Flag: This group of sixteen bits is used for flags such as Internal Busy (the device has a pending call currently), no-wait mode, Device Online, and Device.

Handle: This long-word value is the handle associated by the memory manager with this memory segment. Any call to the memory manager concerning this segment will be referenced by this handle. Normally this will be when this segment is disposed of as per the discussion for the Memory DIB Count above.

Block Size: The data for this four byte field is also taken from the MODE SENSE call to the device through the SCSI Manager. This information is three bytes long and will be in bytes five through seven of the *Block Descriptor* field of the returned data. It will also be in a High to Low format and is stored in the DIB extension.

Max Command: This word value is the maximum command accepted by this device. SCSI has allowed for commands in the range of $00-$FF, but it may be that this device will accept no command greater than $25, for example. There is no need to waste the user's time if we know the command will not be accepted.

Command Bitmaps: The DIB extension data is built from the data returned in the INQUIRY call. These data can not simply be moved in. They must be handled on a group by group basis. Not all groups will be represented in the return data. This will be explained in more detail in the next section.

SCSI Manager Command Area: The following several bytes are used as work space for the command structures sent to the SCSI Manager.

Completion Vector: This section contains code that when called will locate itself and clear an internal busy flag for this device when an asynchronous completes. Until this flag is cleared, no other commands to this device will be accepted. If sent, the driver will wait for the busy bit to clear rather than return an error to the caller.

Building the DIBs in three short SCSI Calls.

Above we briefly described how to build the DIB on a field by field basis. Not much detail was given because of the inefficiency of that method. Here we will describe in detail the calls to be made to the SCSI Device and show what data fields to expect in return.

Besides the Get_Devices call that we have already issued, we will need to make three additional calls to the device. To get all the device information that we need to build the DIB we need to issue the INQUIRY, MODE SENSE and READ CAPACITY commands. We will start with the INQUIRY Command.

INQUIRY: In **Figure 8** we have the command block for the first of the calls that we need to issue. This is the INQUIRY call and is covered in the ANSI® X3.131-1986 SCSI document starting on page 69. The only field that we need to worry about is the *Allocation Length*. If this is zero then none of the important values will be returned. To account for any anomalies between devices, it might be wise to set this to $FF.

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code $12							
1	Logical Unit #							
2	Reserved							
3	Reserved							
4	Allocation Length							
5	Vendor Unique		Reserved				Flag	Link

Figure 8

In response to this call the device will return the structure shown in **Figure 9**. Although we need only a couple of fields from this to finish off our DIB, it might be prudent to check some of the other fields also. Byte zero is the *Peripheral Device Type*. This can be checked to ensure that this is the type of device that the driver wishes to converse with.

Byte one bit 7, if set to 1, indicates that the media is removable.

Bytes $05 - $25 can be used to verify a particular vendor or model of the device. The revision number can also be checked here. These could have meaning to a driver that is written for a specific vendor's product.

Starting at Byte $26 is the start of the command bitmaps. These are arranged as 5 byte sets. The first byte is the group number and will be in the range of 0 - 7. An $FF indicates that there are no more command groups to follow. The next 4 bytes are the bitmap itself. Byte 0 Bit 7 = command 0, bit 0 = command 7, Byte 1 bit 7 = command 8 and so on. These fields will need to be moved individually into the DIB extension to

allow for holes in the bitmap. This is because the device may only return commands for group 0 and group 6 with nothing in between.

Bit / Byte	7	6	5	4	3	2	1	0
0	Peripheral Device Type							
1	RMB	Device-Type Qualifier						
2	ISO Version		ECMA Version			ANSI-Approved Ver		
3	Reserved							
4	Additional Length							
	Vendor Unique Parameters							
5	Vendor Unique							
6-7	Reserved							
8-F	Vendor ID in ASCII							
10-1F	Product ID in ASCII							
20-23	Revision Level in ASCII							
24-25	Number of Resrve/Release extents							
26	Group Number							
27-2A	Group Commands Implemented							
2B	Group Number							
2C-2F	Group Commands Implemented							
n	$FF end of commands Implimented							

00h = Direct Access Device
01h = Sequential Access Device
02h = Printer Device
03h = Processor Device
04h = WORM Device
05h = ROM Direct Access Device (CD)
06h-7Eh = Reserved
7Fh = Logical Unit not present
80h-FFh = Vendor Unique

See SCSI Spec. Page 70

Figure 9

MODE SENSE: This command is covered in the ANSI® X3.131-1986 SCSI document starting on page 108. This is a 6 byte command from the group 0 list of commands. The parameter or command list is shown in **Figure 10.**

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code $1A							
1	Logical Unit #							
2	Reserved							
3	Reserved							
4	Allocation Length							
5	Vendor Unique		Reserved				Flag	Link

Figure 10

In response to this call the data structure shown in **Figure 11** will be returned into your buffer. From this information you can determine whether or not the device is write protected and the size of each blocks in bytes, and the block count. Not all

devices return this block data. Block device drivers should use the READ CAPACITY Call. That is all that we need from this call. It should be noted that even though this call does return the block count for this device, (if a block device) we will not be using this data.

Byte \ Bit	7 6 5 4 3 2 1 0	
0	Sense Data Length	Total Length of all Data Returned
1	Medium Type	
2	WP (bit 7); Reserved	WP = Write Protected Bit (1 = Protected)
3	Block Descriptor Length	
	Block Descriptor(s)	
0	Density Code	\$00 Default (Only one density supported \$01 Flexible disk, single density \$02 Flexible disk, double density \$03 - \$7F Reserved \$80 - \$FF Vendor Unique
1	Number of Blocks (MSB)	Number of Blocks on the device
2	Number of Blocks	
3	Number of Blocks (LSB)	
4	Reserved	
5	Block Length (MSB)	Block Size in Bytes
6	Block Length	
7	Block Length (LSB)	
	Vendor Unique Parameter(s)	
0 -N	Vendor Unique Parameter Byte(s)	

Figure 11

READ CAPACITY: This command is also covered in the ANSI® x3.131-1986 and SCSI-2 document. This is a 10 byte command from the group 1 list of commands. The parameter or command list is shown in **Figure 12.**

Byte \ Bit	7	6	5	4	3	2	1	0
0	Operation Code $25							
1	Logical Unit #							
2	Logical Block Address (MSB)							
3	Logical Block Address							
4	Logical Block Address							
5	Logical Block Address (LSB)							
7	Reserved							
8	Allocation Length							
9	Reserved							
A	Vendor Unique		Reserved				Flag	Link

Figure 12

In response to this call the data structure shown in **Figure 13** will be returned into your buffer. The READ CAPACITY Call is used in the case of a block device to find the total size of the device in blocks and also the size of the block in bytes. The block number given is the last readable block on the disk. To get the total block count, the block number is incremented by one. This is to account for block zero being the first block. This information is also supplied by the MODE SENSE Call but not by all devices, so rather than take a chance on the values, we will use this call for this information.

Byte \ Bit	7	6	5	4	3	2	1	0
0	Logical Block Address (MSB)							
1	Logical Block Address							
2	Logical Block Address							
3	Logical Block Address (LSB)							
4	Block Length (MSB)							
5	Block Length							
7	Block Length							
8	Block Length (LSB)							

Figure 13

Reading the Partitions: Block devices can be partitioned into several smaller logical volumes. These volumes can vary is size and structure or Operating System. The Operating System type will be up to the FSTs to manage. The size is our responsibility. Using the READ CAPACITY Call, the driver knows the total size of this device. If partitioned, this is no more than the sum of all the partition sizes. The driver must therefore read the Partition Map and build a DIB for each of the partitions found except for those of the types 'Apple_partition_Map', 'Apple_Scratch', and 'Apple_Free' (these strings are not case sensitive).

The structure of the Partition Map is given in the section for STATUS Calls.

When partitioned media is encountered, the driver will need to ensure that the Head Link and Forward Device Link pointers are set correctly. There is also a bit in the device characteristics word of the DIB that must be set. The driver must also ensure that the block count matches the one from the Partition Map and not the READ CAPACITY block count.

Driver Open Call

Call Parameters :	Device Number	≠	$0000
	Call Number	=	$0001
	DIB Pointer		

Device Number: *This word parameter specifies the target device. This parameter must be nonzero.*

Call Number: *This word parameter specifies the type of call.*

DIB Pointer: *This longword points to the DIB for the target device.*

The Driver_Open call is used to open a character device for data I/O. If this call is received by a driver written for a block device, it should take no action and return to the caller with no error.

In the case of character devices, this call should be handled on a device by device basis. Some devices would use the *LOAD/UNLOAD* commands. Other devices might use the *START/STOP Unit* calls. This can be any call or series of calls that ensure that the device is online and ready to respond to the users requests.

The driver should maintain a flag to indicate whether or not a previous open has been received without an accompanying close. If two opens are received the driver returns an **ALREADY_OPEN** error.

Driver Read Call

Call Parameters :	Device Number	≠	\$0000
	Call Number	=	\$0002
	Buffer Pointer		
	Request Count		
	Transfer Count		
	Block Number		
	Block Size	(Char. = \$0000, Block ≠ \$0000)	
	FST Number	(Block device only)	
	Cache Priority	(Block device only)	
	Volume ID	(Block device only)	
	DIB Pointer		

Device Number: *This word parameter specifies the target device. This parameter must be nonzero.*

Call Number: *This word parameter specifies the type of call.*

Buffer Pointer: *This longword points to where the data is to be returned.*

Request Count: *This longword specifies the number of bytes requested from the target.*

Transfer Count: *This longword indicates the number of bytes actually transferred.*

Block Number: *This longword parameter is the logical block address from which data is to be read from. This parameter has no application in character device drivers.*

Block Size: *This word parameter specifies the size of the block addressed by block number. This parameter must be a nonzero value for block devices or a zero for character devices.*

FST Number: *This word parameter specifies the File System Translator that owns the volume from which the block is being read. When set, the most significant bit of the FST number will force device access during the read even if the block being accessed is in the cache. In this case no cache access will occur. If a zero, then the call came from a device call from the application.*

Cache Priority: *This word parameter specifies whether or not caching is to be invoked for the block specified in the current I/O transaction. A value of \$0000 specifies that the block is not to be placed into the cache. A cache enable of \$0001 through \$7FFF implements caching using an LRU algorithm. If no space is available to cache the block, the*

least recently used purgable block will be purged to make room for caching the block in the current I/O request. A cache enable of $8000 through $FFFF also use the LRU algorithm but will be cached as deferred unpurgable blocks. Non-deferred cached blocks are cached by device number while deferred cached blocks are cached by volume ID. Read operations do not invoke deferred caching. The device dispatcher limits cache priorities to the range of $0000 to $7FFF during read operations. This parameter has no application in character device drivers.

Volume ID: *This word is used to identify deferred cached blocks belonging to a specific volume.*

DIB Pointer: *This longword points to the DIB for the target device.*

This call is supported by both character and block device drivers. It translates into a SCSI Read Command. If the driver is for a character device then it must have an active **Opened** device. If an I/O transaction is attempted with a device that has not been opened, the driver will return a 'NOTOPEN' error. Block devices do not have this requirement. In either case, the transfer count will be set to zero prior to any calls to the device and then updated by the driver as data is received. A character device should also verify that the block size is set to zeros. If the size is non-zero, a 'NOT A BLOCK DEVICE' error shall be returned.

Block devices also need to check that the block size is correct for the targeted device. If the block size is incorrect it will return a 'BAD CALL PARAMETER', or if the request count is not an integral multiple of block size, the driver should return an 'INVALID BYTE COUNT' error. If the block number (range if Request Count is greater than Block Size) is not within the legal range, a 'BAD BLOCK' error should be the response. As blocks are read from the device, the transfer count will be incremented by block_size bytes until Request Count bytes have been transferred. If the blocks are read as part of a multi-block request, the count will be updated at the end of that request.

Driver Write Call

Call Parameters :	Device Number	≠	$0000
	Call Number	=	$0003
	Buffer Pointer		
	Request Count		
	Transfer Count		
	Block Number		
	Block Size	(Char. = $0000, Block ≠ $0000)	
	FST Number	(Block devices only)	
	Cache Priority	(Block devices only)	
	Volume ID	(Block devices only)	
	DIB Pointer		

Device Number: *This word parameter specifies the target device. This parameter must be nonzero.*

Call Number: *This word parameter specifies the type of call.*

Buffer Pointer: *This longword points to where the data is to be written from.*

Request Count: *This longword specifies the number of bytes requested to send.*

Transfer Count: *This longword indicates the number of bytes sent to the target.*

Block Number: *This longword parameter is the logical block address to which data is to be written to. This parameter has no application in character device drivers.*

Block Size: *This word parameter specifies the size of the block addressed by block number. This parameter must be a nonzero value for block devices or a zero for character devices.*

FST Number: *This word parameter specifies the File System Translator that owns the volume to which the block is being written. The most significant bit of the FST number has no effect on a write call. If zero, then the call came from the application via a device call.*

Cache Priority: *This word parameter specifies whether or not caching is to be invoked for the block specified in the current I/O transaction. A value of $0000 specifies that the block is not to be placed into the cache. A cache enable of $0001 through $7FFF implements caching using an LRU algorithm. If no space is available to cache the block, the least recently used purgable block will be purged to make room for caching the block in the current I/O request. A cache enable of $8000 through $FFFF also use the LRU algorithm but will be cached as deferred unpurgable blocks. Non-deferred cached blocks are cached by device number while deferred cached blocks are cached by volume ID. This parameter has no application in character device drivers.*

Volume ID: *This word is used to identify deferred cached blocks belonging to a specific volume.*

DIB Pointer: *This longword points to the DIB for the target device.*

This call is supported by both character and block device drivers. It transfers data from the buffer specified in the parameter block on direct page to the target device. Like the Driver Read Call, this will translate into a SCSI Write command regardless of whether the device is a block or character device.

If a character device is the target, the driver shall verify that the device has been and still is opened. If the device is not opened, the driver will return a 'NOT_OPEN' error. The block size should be set to zero or a 'NOT_A_BLOCK_DEVICE' error will be returned. As data is written to the device the transfer count will be incremented. This count should have been set to zero on entry to this call.

When the target device is a Block device the Block Size shall be validated. If it is incorrect it will return a 'BAD CALL PARAMETER', or if the request count is not an integral multiple of block size, the driver will return an 'INVALID BYTE COUNT' error. If the block number (range if Request Count is greater than Block Size) is not within the legal range, a 'BAD BLOCK' error will be the response. The driver resets the transfer count to zero and as data is written the count will be incremented by Block Size for every block read. If the driver reaches the end of the disk before request count blocks but after at least one block is read then no error shall be returned and the transfer count will reflect correctly the amount returned. If the blocks are read as part of a multi-block request, the count will be updated at the end of that request.

Driver Close Call

Call Parameters :	Device Number	≠	\$0000
	Call Number	=	\$0004
	DIB Pointer		

Device Number: *This word parameter specifies the target device. This parameter must be nonzero.*

Call Number: *This word parameter specifies the type of call.*

DIB Pointer: *This longword points to the DIB for the target device .*

This call is the counter to the **Driver_Open** Call and is used when talking to character devices. On receiving this call, the driver should return to the same state that was in effect when the **Driver_Open** call was issued. All memory from the Memory Manager is released, the internal driver open flag is set to not open. If the driver was not in the open state when this call was issued a **NOT_OPEN** Error is returned.

If the Driver is a Block Device Driver no action will be taken and the driver returns no error.

Driver Status Call

Call Parameters :	Device Number	≠	$0000
	Call Number	=	$0005
	Status List Pointer		
	Request Count		
	Transfer Count		
	Status Code		
	DIB Pointer		

Device Number: *This word parameter specifies the target device. This parameter must be nonzero.*

Call Number: *This word parameter specifies the type of call.*

Status List Pointer: *This longword points to where the status list is to be returned.*

Request Count: *This longword indicates the number of bytes to be returned. If the request count is smaller than the minimum buffer size required by the call, an error is returned.*

Transfer Count: *This longword indicates the number of bytes actually transferred.*

Status Code: *This word parameter specifies the type of status request. Status codes of $0000 through $7FFF are standard calls that must be supported by device drivers. SCSI specific status calls use status codes in the range of $8000 through $FFFF.*

The list of standard status calls are as follows:

$0000	Return Device Status
$0001	Return Configuration Parameters
$0002	Return Wait/No Wait Status
$0003	Return Format Options
$0004	Return Partition Map
$0005	Return Last Command Result
$0006 - $7FFF	Reserved - These status codes to be assigned by Apple Computer, Inc.

The list of device specific status calls are as follows:

$8000 - $80FF	Device specific SCSI commands.

The following are the SCSI specific status calls. The values to the right are the Devices that use that code and are defined at the end of the list.

Group 0 Commands

$8000	Test Unit Ready	0,A,B,C,D,E
$8003	Request Sense	0,A,B,C,D,E
$8005	Read Block Limits	2
$8006	Receive QIC-100 System Data	E
$8008	Read	1,2,5,6,8,B
	Receive	4
	Get Message	A
$800C	Reserved	
$800D	Read SCSI Defect Data	E
$800E	Read Controller Information	E
$800F	Read Reverse	2
$8011	Read Drive Lines	E
$8012	Inquiry	0,A,B,C,D,E
$8013	Read QIC-100 Information	E
$8014	Recover Buffered Data	2,3
$8019	Read QIC-100 Defect Data	E
$801A	Mode Sense	1,2,3,5,6,7,8,A,B,C,D,E
$801C	Receive Diagnostic Results	0,A,B,E
$801F	Read Log	2

Group 1 Commands

$8020 - $8023	Reserved	
$8025	Read Capacity	1,5,6,8,B,E
	Get Window Parameters	7
$8026 - $8027	Reserved	
$8028	Read	1,5,6,7,8,B,C,E
$8029	Reserved	
$802D	Read Update Block	8
$8034	Read Position	2
	Get Data Status	7,C
$8037	Read Defect Data	1,E
$8038	Read Element Status	9
$803C	Read Buffer	1,2,7,8,A,B,E
$803E	Read Long	1

Group 2 Commands

$8040 - $8059	Reserved	
$805A	Mode Sense	0
$805B - $805E	Reserved	
$805F	Read Log	0

Group 3 Commands

\$8060 - \$807F	Reserved	

Group 4 Commands

\$8080 - \$809F	Reserved	

Group 5 Commands

\$80A0 - \$80A4	Reserved	
\$80A7	Reserved	
\$80A8	Read	8
\$80A9	Reserved	
\$80AB	Reserved	
\$80AD	Read Update Block	8
\$80B4 - \$80B6	Reserved	
\$80B7	Read Defect Data	8
\$80B8 - \$80BC	Reserved	
\$80BE - \$80BF	Reserved	

Group 6 Commands

\$80C1	Read TOC	B
\$80C2	Read Q Subcode	B
\$80C3	Read Header	B
\$80C4 - \$80C7	Reserved	
\$80CC	Audio Status	B
\$80CE - \$80DF	Reserved	

Group 7 Commands

$80E0 - $80FF Reserved

Device Definitions

0	=	Devices '1' through '9'
1	=	Direct-Access Devices
2	=	Sequential-Access Devices
3	=	Printer Devices
4	=	Processor Devices
5	=	Write-Once Read-Multiple Devices
6	=	Read-Only Direct-Access Devices
7	=	Scanner Devices
8	=	Optical Memory Devices
9	=	Changer Devices
A	=	Communications Devices
B	=	Apple CD_ROM Drive
C	=	Apple SCSI Scanner
D	=	Apple LaserWriter SC
E	=	Apple Tape Drive

Non-SCSI Commands

$8100 - $FFFF Reserved

DIB Pointer: *This longword points to the DIB for the target device.*

This call is used to obtain current information from the device or driver itself and may be used to issue any extended SCSI Command that returns data from the device through the use of device specific control codes. The device driver is responsible for validating the status code prior to executing the requested command. The following 65816 code sample shows how this might be done.

```
validate_stat_code      lda     status_code             ;Get Status Code
                        bmi     device_specific         ;Is it Device Specific?
                        cmp     #max_command+1          ;No. Is it out of range?
                        blt     do_standard             ;No. Goto code segment
bad_code_exit           lda     #BAD_CODE               ;Central BAD CODE Exit
                        sec
                        rtl

device_specific         and     #$00ff                  ;Is it ≤ the max SCSI
                        pha                             ;command for this
                        inc                             ;device?
                        ldy     #SCSI_maxcmd_offset
                        cmp     [DIB_PTR],y
                        blt     so_far_so_good          ;It's in a good range
                        pla                             ;Restore stack
                        bra     bad_code_exit           ;exit with error

so_far_so_good          lda     1,s                     ;Get status code from stack
```

```
                lsr                              ;Generate a group index
                lsr                              ;to use to get to the
                lsr                              ;correct group offset
                lsr
                and     #$000e
                tay
                lda     group_offset,y
                sta     temp                     ;Save offset for later

                lda     1,s                      ;Get status code from stack
                and     #$0010                   ;Upper or lower half
                beq     first_16_bits            ;of group bitmap?
                inc     temp                     ;Upper half. Adjust
                inc     temp                     ;offset by two

first_16_bits   pla                              ;Last time. Get status code
                and     #$000f                   ;Retain low nibble
                asl     a                        ;Account for 16 bit
                tay                              ;table of offsets.
                lda     cmd_bm_mask,y
                ldy     temp
                and     [DIB_PTR],y              ;Is bit set in bitmap?
                beq     bad_code_exit            ;No. Error exit.
                        .
                        .
                        .
max_command     equ     $0004                    ;Max standard status code

SCSI_maxcmd_offset equ  $0042

group_offset    dc      i2'GROUP0-DIB_start'     ;Offsets from start
                dc      i2'GROUP1-DIB_start'     ;of DIB to Group
                dc      i2'GROUP2-DIB_start'     ;bitmaps
                dc      i2'GROUP3-DIB_start'
                dc      i2'GROUP4-DIB_start'
                dc      i2'GROUP5-DIB_start'
                dc      i2'GROUP6-DIB_start'
                dc      i2'GROUP7-DIB_start'

temp            ds      2

cmd_bm_mask     dc      i2'%1000000000000000'    ;Bitmap masks.
                dc      i2'%0100000000000000'
                dc      i2'%0010000000000000'
                dc      i2'%0001000000000000'
                dc      i2'%0000100000000000'
                dc      i2'%0000010000000000'
                dc      i2'%0000001000000000'
                dc      i2'%0000000100000000'
                dc      i2'%0000000010000000'
                dc      i2'%0000000001000000'
                dc      i2'%0000000000100000'
                dc      i2'%0000000000010000'
                dc      i2'%0000000000001000'
                dc      i2'%0000000000000100'
                dc      i2'%0000000000000010'
                dc      i2'%0000000000000001'
```

If an invalid status code is passed to the driver, the driver will return a 'BAD CODE' error. The driver may also wish to identify where the call came from and shield certain calls from the application. Based on this, the driver could use a secondary command bitmap to validate codes allowed from the application. If an invalid control list length is passed to the driver, the driver should return a 'BAD PARAMETER' error. The device driver sets the transfer count to the number of bytes returned as a result of the status call.

NOTE: Both standard and device specific status calls may detect an OFFLINE or DISKSWITCH status. If either of these conditions are detected then a **SET_DISKSW** call is issued to set the device dispatcher maintained disk switched error.

$0000 - Return Device Status

This call returns a general status followed by a longword specifying the target device in blocks. The block count for character devices will be returned as zero. The general status word indicates the status of the device.

The bit definition within the general status word for character devices is as follows:

Bit 15	Reserved (currently read as zero)
Bit 14	1 = Linked Device
Bit 6-13	Reserved (currently read as zero)
Bit 5	Buffer not empty
Bit 4	1 = Online, 0 = Offline
Bit 2-3	Reserved (currently read as zero)
Bit 1	1 = Device currently interrupting
Bit 0	1 = Device currently open

The bit definition within the general status word for block devices is as follows:

Bit 15	1 = Block count is uncertain for current block size
Bit 14	1 = Linked Device
Bit 5-13	Reserved (currently read as zero)
Bit 4	1 = Disk in drive, 0 = Disk not in drive
Bit 3	Reserved (currently read as zero)
Bit 2	1 = Write protected, 0 = Write enabled
Bit 1	1 = Device currently interrupting
Bit 0	1 = Disk has been switched

If either bit 0 or bit 4 has been set then the driver will call the SET_DISKSW via the system service call table.

Disk switched status should be set on ejection for optimum performance. A status call should only return an error code if the status call fails. Error codes should not be returned for conditions indicated with the general status word.

Status List:	Word	General status word
	Long	Number of blocks supported by device

$0001 - Return Configuration Parameters

This call returns a byte count as the first word in the status list which indicates the length of the configuration parameter list in bytes. The configuration parameters will be placed into the status list contiguous to the byte count. The structure of the configuration parameter list is device dependent.

Status List:	Word	Length of configuration parameter list
	Data	Returned configuration parameters

$0002 - Return Wait/No Wait Status

This call is used to determine if a device is in wait mode. If in wait mode, the device will wait for the number of characters specified in the request count of a read call before returning from the read. If in No Wait mode, a read call will return immediately with a transfer count indicating the number of characters returned. If a character was available the transfer count will return from a read with a non zero value. If a character was not available the transfer count will return from a read call with a value of zero. A word with a value of $0000 returned in the status list indicates that the device is operating in Wait mode. A word with a value of $8000 returned in the status list indicates that the device is operating in No Wait mode. Block devices always operate in Wait mode.

Status List:	Word	Wait/No Wait status

$0003 - Return Format Options

This call returns a list of formatting options that may be selected using a Set_Format_Options call prior to issuing a format call to a block device. These parameters may include such variables as format environment, number of blocks, block size, and interleave. Devices that do not support media variables will return with a transfer count of zero and no error. The format of the status list on return from this call when a device does support media variables is as follows:

Returned List:	Word	Number of entries in list
	Word	Number of displayed entries in list
	Word	Recommended Default Option
	Word	Option that current online media is formatted with

Then each entry in the list consists of 16 bytes containing the following 5 fields:

Word	Media variables reference number
Word	Reference number of linked entry
Word	Flags
Long	Number of blocks supported by device
Word	Block Size
Word	Interleave Factor
Word	Media size (block count * block size)

Flags word definition is as follows:

Bits 0 - 1	Format Type
Bits 2 - 3	Size Multiplier
Bits 4 - 15	Reserved (must be zero)

Format Type definition is as follows:

00	Universal Format
01	Apple Format
10	Non-Apple Format
11	Not valid

Size Multiplier definition is as follows:

00	Size in bytes
01	Size in kilobytes
10	Size in megabytes
11	Size in gigabytes

A typical list returned from this call for a device supporting two possible interleaves intended to support Apple's file systems (ProDOS, GS/OS, MFS or HFS) might be as follows:

Transfer count	$00000038	56 bytes returned in list
Returned List:	$0003	Three entries in list
	$0002	Only two display entries
	$0001	Recommended default is option #1
	$0003	Current media is formatted as specified by option #3
	$0001	Refnum = Option #1
	$0002	LinkRef = Option #2
	$0008	Universal format / size in megabytes
	$0000A2F9	Block count = 41721
	$0200	Block size = 512 bytes
	$0001	Interleave factor = 1:1
	$0020	Media size = 20 megabytes
	$0002	Refnum = Option #2
	$0000	LinkRef = none
	$0008	Universal format / size in megabytes
	$0000A2F9	Block count = 41721
	$0200	Block size = 512 bytes
	$0002	Interleave factor = 2:1
	$0020	Media size = 20 megabytes

$0003	Refnum = Option #3
$0000	LinkRef = none
$0008	Universal format / size in megabytes
$00009C8C	Block count = 40076
$0214	Block size = 532 bytes
$0005	Interleave factor = 5:1
$0019	Media size = 19 megabytes

Character devices should return no error.

$0004 - Return Partition Map

This call returns the partition map of the device specified which must be the first device of any linked list. If not an **Invalid Device Number** error is returned.

The size of the partition map can vary and the only way for an application to be sure that the entire partition map has been read is to validate the pmMapBlkCnt field in the map with the transfer count returned by the driver. If pmMapBlkCnt * block size ≠ Transfer Count then the application must reissue the call with the appropriate request count. If Request Count is not an integral multiple of block size, then an **Invalid Byte Count** error is returned. All calls requesting the partition map will return the data starting with the first entry. There is no allowance for reading a specific entry in the partition map. If the application wishes to reassign an entry to a different type OS, the $0006 control call Assign Partition Owner should be used.

The structure of the partition map is described in detail in '**Inside Macintosh™** Volume V' under the SCSI Manager section. Below (**Figure 14**) is a picture of what a partition map entry looks like along with definitions of a few of the fields. It must be noted that all fields, except strings, in the partition map are stored High Byte »» Low Byte. Example: Even though the partition signature is defined as $504d, it will appear to the 65xxx family of processors as $4d50.

byte	Field	Description
0	pmSig (word)	always $504d
2	pmSigPad (word)	reserved for future use
4	pmMapBlkCnt (long word)	number of blocks in map
8	pmPyPartStart (long word)	first physical block of partition
C	pmPartBlkCnt (long word)	number of blocks in partition
10	pmPartName (32 bytes)	partition name
30	pmPartType (32 bytes)	partition type
50	pmLgDataStart (long word)	first logical block of data area
54	pmDataCnt (long word)	number of blocks in data are
58	pmPartStatus (long word)	partition status information
5C	pmLgBootStart (long word)	first logical block of boot code
60	pmBootSize (long word)	size in bytes of boot code
64	pmBootLoad (long word)	boot code load address
68	pmBootLoad2 (long word)	additional boot load information
6C	pmBootEntry (long word)	boot code entry point
70	pmBootEntry2 (long word)	additional boot entry information
74	pmBootCksum (long word)	boot code checksum
78	pmProcessor (16 bytes)	processor type
88	(128 bytes)	boot specific arguments

Figure 14

The only fields that shall be defined here are the pmPartType (Partition Type) and the pmProcessor (Processor Type). These are ASCII strings of 1 to 32 or 16 bytes respectively in length; case is not significant. If either name is less than the max character length, it must be terminated with a NULL character (ASCII code 0). An empty name can be specified by setting the first byte to the NULL character.

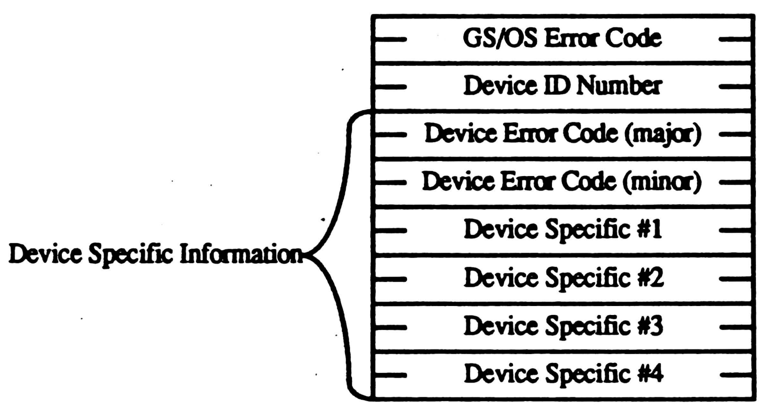
Example Types	Example Processors			
Apple_MFS	68000	8080	80186	6502
Apple_HFS	68008	8085	80286	65C02
Apple_Unix_SVR2	68010	Z80	80386	65802
Apple_partition_map	68012	8086	80486	65816
Apple_Driver	68020	8088		
Apple_PRODOS	68030	Z8000		
Apple_Free	68040	6800		
Apple_Scratch		6809		

$0005 - Return Last Command Result

This command is used to query the driver for the results of the last command issued. This is used in the case of an Async command to verify that it did perform the requested action. (Example. An Async Format call is issued to the device that may take several minutes. The command may have returned no error up front, but then have problems performing the actual Format of the media. In this way the application could be aware that the media is defective and warn the user).

This call would return request length bytes up to 16 bytes maximum. The returned information would be structured as below.

> The first word would be from the set of GS/OS error codes and would reflect the problem encountered while performing the last requested transaction as closely as possible.
>
> The second word would be the Device ID Number. This would aid in deciphering the device specific information that follows.
>
> The next 12 bytes would be the device specific information pertaining to the device in question. (This information will be null if the request was successfully completed).



In the world of SCSI, the device specific information would be as pictured below and the 12 bytes of device specific information are as follows:

	GS/OS Error Code	
	Device ID Number	
Device Specific Information	Device Error Code (major)	Status Code returned by the SCSI Device
	Device Error Code (minor)	Sense Key and other information
	Device Specific #1	Information Bytes High Word (MSB»»LSB)
	Device Specific #2	Information Bytes Low Word (MSB»»LSB)
	Device Specific #3	Data Length
	Device Specific #4	Extended Sense Key

Device Error Code (major) - This is the status code returned from the target device during the Status Phase and are as defined in the following list.

rr00000r	=	Good. No Error. ('r' = Reserved)
rr00001r	=	Check Condition
rr00010r	=	Condition Met/Good
rr00100r	=	Busy
rr01000r	=	Intermediate/Good
rr01010r	=	Intermediate/Condition Met/Good
rr01100r	=	Reservation Conflict
rr10100r	=	Queue Full

All others are reserved.

Device Error Code (minor) - This is a combination of several bits and the Sense Key. These bits and key are defined as follows.

Bit 7	=	1 indicates that the command read a filemark. This is only used for sequential access devices.
Bit 6	=	1 indicates an end of medium condition. It is used in sequential access and printer devices. This includes End-of-tape, beginning-of-tape, out-of-paper, ect. Not used in direct access devices.
Bit 5	=	1 indicates that the requested logical block length did not match the logical block length of the data on the media. (Please refer to the ANSI SCSI Spec for a discussion of logical block lengths).
Bit 4	=	Reserved.

Bit 3-0 = Sense Key.

0	=	No Sense
1	=	Recovered Error
2	=	Not Ready
3	=	Medium Error
4	=	Hardware Error
5	=	Illegal Request
6	=	Unit Attention
7	=	Data Protect
8	=	Blank Check
9	=	Vendor Unique
A	=	Copy Aborted
B	=	Aborted Command
C	=	Equal
D	=	Volume Overflow
E	=	Miscompare
F	=	Reserved

Device Specific #1 and #2 - These are used to form a Long result ordered from MSB to LSB and contain information about the failure. A few values will be discussed briefly and the reader is urged to read the ANSI SCSI Spec for further details concerning this information.

(1) The unsigned block address associated with the sense key.

(2) The difference (residue) of the requested length minus the actual length in either bytes or blocks, as determined by the command. (Negative values are indicated by two's compliment notation).

Device Specific #3 - The length of the data returned if the next command is a Request Sense Command requesting further data concerning the error.

Device Specific #4 - This will contain the extended Sense Key that was returned by the device and defined in the ANSI SCSI Spec as well as the Device Vendors own documentation.

Device Specific Status Calls

The remaining Status Code definitions pertain to the SCSI Device specific commands. To issue a single command, the caller will use the parameter block structure (**Figure 15**) below The Buffer Pointer on direct page points to this structure and the REQUEST COUNT, also on direct page, is the number of bytes total requested by this command.

(NOTE: The request count should not be larger that what is allowed by the device for the call being issued.)

The first word is the call version number. To maintain compatibility with previous drivers written for the GS/OS Operating System, we need to distinguish between Versions or class of SCSI Device specific commands. The first release supported

version $0000 only. Under version $0000, only the version number itself, the Command Specific Data and the buffer pointer were included. In this version of the driver, we will additionally support version $0001.

The next field is the twelve byte command area and contains the information as outlined in the definition of the command that is being issued. This field is present in all versions.

The Buffer Pointer is where the data for the call is to be found at, returned to, depending on the call being sent. This field is present in all versions.

The remaining fields are supported by version $0001, and are discussed in detail, along with the previous two fields and how they work together, in the text below.

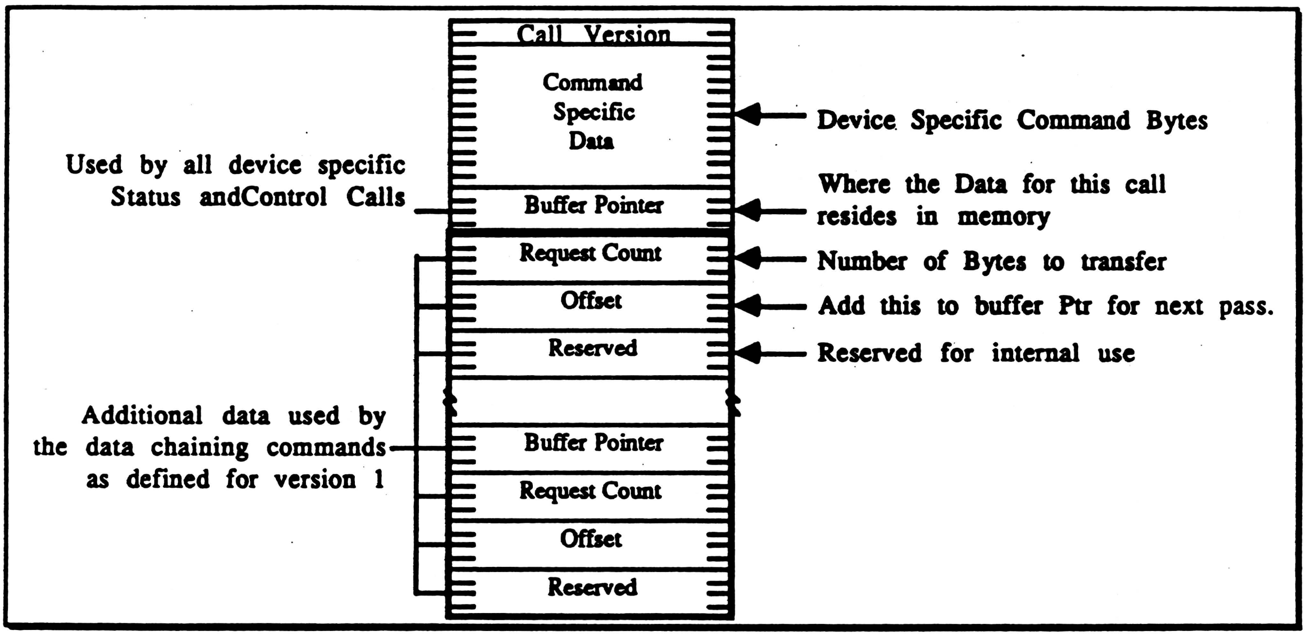


Figure 15

There are times when a transaction must be broken up into it's separate components. In these situations the Application might choose to use the data chaining capabilities of the SCSI Driver/Manager. These data chaining commands are each composed of four long word fields and are in the normal Low »» High format.

There are three commands that compose the data chaining structure. By using them together in various combinations data can be gathered or distributed depending on the direction of transfer. The three commands are listed below:

- The first is a value in the range of $00000001 - $ffffffe. Any value in this range will be treated as a buffer address. This is referred to as the DCMove instruction.
- The second command is $ffffffff. This is the DCLoop Command.
- The third, $00000000, is the DCSpecial Command.

Depending on what is in the Buffer field, the next field will be interpreted as given below.

- If a DCMove instruction, then this is a requested byte count.
- This is interpreted as a count for the number of times to loop through the buffer entries for the DCLoop Command.
- In the case of a DCSpecial Command, this field is used by the application to either stop the sequence of Data Chaining Commands ($00000000) or an address of a routine to call if the field is non-zero (<>$00000000). This can be used by the application to indicate the action to be taken such as a graphics page flip or some other task that would need to be performed mid transfer.

The third field also has three definitions depending on the contents of the buffer address. The are as follows.

- Offset to add to the buffer address for the next pass for DCMove.
- Number of entries to go back to restart the list when DCLoop.
- Reserved and should be null for the DCSpecial STOP instruction. If not a STOP Command, then this can contain flags to indicate where in the data chaining command structure the SCSI Manager is currently pointing.

The fourth field is for internal usage and must be null in all cases.

(NOTE: The request count times the loop count plus any other request counts may not be larger than the REQUEST COUNT used when issuing the call. If it is larger, a time-out error will be returned, the data chaining structures will have been modified and the data returned is unreliable.)

Two examples for using these commands are given below. One for receiving data and the other for sending. It should be noted that there is no support implied for the devices used in the following examples.

Sending Data:

A printer is connected to the SCSI interface that requires a header or declaration to be sent prior to sending the image to be printed. The image resides in memory as contiguous pages and is rather large. Each page of data is the same size.

Rather than breaking up the data to be printed and shuffling it with copies of the header information, the Application could instead use the following data chaining commands.

DCMove	$00000200	$00000000	$00000000
DCMove	$00004000	$00004000	$00000000
DCLoop	$00000019	$fffffffe (-2)	$00000000
DCStop	$00000000	$00000000	$00000000

The first DCMove is the pointer to the header information that will be sent each time through the loop. It is 512 bytes in size and we want to sent the same data on each pass of the loop.

The second DCMove points to the first image to be sent. Each page is 16 KB in size and we want to send the next page which starts right after the current one on the next pass of the loop.

DCLoop tells us to loop $19 times and to go back 2 instructions to the first DCMove each time. When the loop executes the $19th pass it will drop through to the next command which is the DCStop. This will signal the end of the transmission and the call will return via the normal path to the Application.

Receiving Data:

An imaging device is connected to the SCSI interface that will be sending a large image to be displayed. The video mapping as it is in the Apple II family is not structured in a way that allows an image to be read directly in and displayed. The data must be moved around. By structuring the data chaining command, a HIRES Bitmap type Image can be read directly in video memory without having to adjust it first.

Rather than breaking up the Read calls to the device and spending a lot of time in overhead, the Application could instead use the following data chaining commands.

DCMove	$00000028	$00000028	$00000000
DCMove	$00000028	$00000028	$00000000
DCMove	$00000028	$00000028	$00000000
DCMove	$00000028	$00000028	$00000000
DCMove	$00000028	$00000028	$00000000
DCMove	$00000028	$00000028	$00000000
DCMove	$00000028	$00000028	$00000000
DCMove	$00000028	$00000028	$00000000
DCLoop	$00000002	$fffffff8 (-8)	$00000000
DCStop	$00000000	$00000000	$00000000

The first DCMove is the pointer to $0400. This is the first line of video. This will increment by $28 on each pass.

The second DCMove is the pointer to $0480. This is the second line of video. This will increment by $28 on each pass.

The third DCMove is the pointer to $0500. This is the third line of video. This will increment by $28 on each pass.

This continues on to the eighth DCMove which points to $0780.

DCLoop tells us to loop 2 times and to go back 8 instructions to the first DCMove each time. When the loop executes the 3rd pass (1st sequence plus 2 loops = 3 passes) it will drop through to the next command which is the DCStop. This will signal the end of the transmission and the call will return via the normal path to the Application.

Note: Commands are always checked for validity prior to issuance to the SCSI Manager. If any command is unacceptable, the driver returns the error in the Acc. the carry is set.

The list of each Device Specific Status Code follows with each one giving a detailed description of what the Command Data looks like and what data is returned in the buffer as well as what data is needed in the buffer for a few of the calls.

$8000 - Test Unit Ready; (All Devices)
[M]

This command must be accepted by all devices and is used to check if the target device is ready to accept media access commands. This command does not cause the device to do a self test. If no error is returned, the device is ready for access. If the device supports removable media and the media is not inserted, an error is returned. **!!!! Define error handling and error codes to be returned!!!!**

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($00)
	$01	SCSI Command Flags	($00)
	$02 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

Unused.

Transfer Length:

Unused.

Buffer Data Structure:

None.

Errors:

Good
Not Ready

$8003 - Request Sense; (All Devices)
[M]

The Request Sense command must be accepted by all devices and is used to retrieve Sense data from the target device. The sense data is valid for the last command to the target device until the target has received another command. That is this command if it is issued. The data returned is defined by the ANSI® X3.131-1986 and SCSI-2 Specs. **Must request at least 4 bytes.**

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($03)
	$01	SCSI Command Flags	($00)
	$02 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000004 - $000000ff (Bytes) No error if larger

Transfer Length:

$00000004 - $000000ff

Buffer Data Structure:

Sense Data as defined by the ANSI® X3.131-1986 and SCSI-2 Specs are returned in the specified buffer. See also Vendors Device documentation.

Errors:

Good
Check Condition.

$8005 - Read Block Limits; (Sequential Access)
[M]

The Read Block Limits call is aimed at Sequential Access Devices and is mandatory for those devices. It returns the minimum and maximum block size for that device. See the ANSI® x3.131-1986 and SCSI-2 Spec for more details.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($05)
	$01	SCSI Command Flags	($00)
	$02 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

$00000000 - $00000006. A value less than 6 will return an error. A larger value will only return 6 bytes

Transfer Length:

$00000000 if an Error
$00000006 if successful

Buffer Data Structure:

The data returned by this command is shown below. If the maximum length equals the minimum length, then only fixed length blocks of the size indicated are supported. Otherwise the block size can vary between the two ranges. If the max block size = 0 then no upper limit is specified.

$00	Reserved
$01	Maximum Block Len (MSB)
$02	Maximum Block Len
$03	Maximum Block Len (LSB)
$04	Minimum Block Len (MSB)
$05	Minimum Block Len (LSB)

Errors:

Good
Check Condition.

$8006 - Receive QIC-100 System Data; (Apple Tape Drive)

The Receive QIC-100 System Data call is aimed at the Apple Tape Backup Drive only and is mandatory for that device. This command causes the controller to transfer 128 bytes of QIC-100 System Data to the host.

The data that is sent is the contents of the System Data bytes in the buffer memory. If the last command that was executed was a Read command, then the contents of these bytes will be the System Data bytes for the last block read.

See the 3M MCD-40 SCSI Spec for more details.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($06)
	$01	SCSI Command Flags	($00)
	$02 - $0B	Reserved.	

Request Length:

$00000000 - $00000080. A value less than 80 will return an error. A larger value will only return 80 bytes

Transfer Length:

$00000000 if an Error
$00000080 if successful

Buffer Data Structure:

The data returned by this command is shown below.

$00	System Data read with the first
	fram of User Data
$3f	
$40	System Data read with the second
	fram of User Data
$7f	

Errors:

Good
Check Condition.

$8008 - Read; (**Direct Access**)
[M]

This is the standard SCSI Read Command used by most devices that return data to the caller. If a block address greater than $ffff is needed, a command $8028 should be used.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($08)
	$01	Reserved	
	$02 - $03	Logical Block Address	(MSB »» LSB)
	$04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

None.

Errors:

Good
Check Condition.
Reservation Conflict

NOTE: See also **$8008 - Read, Receive and Get Message** described below.

$8008 - Read; (Sequential Access)
[M]

This is the standard SCSI Read Command used by most devices that return data to the caller. If a block address greater than $ffff is needed, a command $8028 should be used.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($08)
	$01	Reserved	(%xxxxxx00)
		SILI	(%000000x0)
		Fixed	(%0000000x)
	$02 - $03	Logical Block Address	(MSB »» LSB)
	$04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

None.

Errors:

Good
Check Condition.
Reservation Conflict

NOTE: See also **$8008 - Read** above, and **Receive** and **Get Message** described below.

$8008 - Receive; (Processor Devices)
[O]

This command is used in the same manner as the read command that shares the Status Code with this command. There are some minor differences though. For further details please refer to the vendors documentation for more details.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($08)
	$01	SCSI Command Flags	($00)
	$02 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $00ffffff (Bytes)

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

Data

Errors:

Good
Not Ready

NOTE: See also **$8008 - Both Reads** above, and **Get Message** described below.

$8008 - Get Message; (Communication Devices) [M]

This command is used in the same manner as the read command that shares the Status Code with these commands. There are some minor differences though. For further details please refer to the vendors documentation for more details.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($08)
	$01	SCSI Command Flags	($00)
	$02 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $00ffffff (Bytes)

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

None.

Errors:

Good
Not Ready

NOTE: See also $8008 - Both Reads and Receive above.

$800D - Read SCSI Defect Data; (Apple Tape Drive)

This command is supported by the Apple Tape Backup Drive but shall be unsupported by any SCSI Driver. The $8037 command shall be the command of choice to receive this information.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($0D)
	$01	Reserved	(%xxxx0000)
		CMPLST	(%0000x000)
		Reserved	(%00000xxx)
	$02 - $0B	Reserved.	

Request Length:

$00000000 - $00ffffff (Bytes)

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

None.

Errors:

Not Supported

$800E - Read Controller Information; (Apple Tape Drive)

This command is supported by the Apple Tape Backup Drive and shall be supported by the SCSI Driver. Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($0E)
	$01	SCSI Command Flags	($00)
	$02 - $0B	Reserved.	

Request Length:

$00000000 - $00ffffff (Bytes)

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this structure.

Errors:

Check Condition

$800F - Read Reverse; (Sequential Access)
[O]

The Read Reverse command is used to optimize for head movement when using a Sequential Access Device. If the head is past the desired block this call would cause the media to move backwards across the head while the data was read. This is more efficient than resetting the head to the beginning of the data and then reading forward again.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($0F)
	$01	Reserved	(%xxxxxx00)
		SILI	(%000000x0)
		Fixed	(%0000000x)
	$02 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $00ffffff (See flag definition)

Transfer Length:

$00000000 - $00ffffff (See flag definition)

Buffer Data Structure:

Data.

Errors:

Good
Check Condition.

$8011 - Read Drive Lines; (Apple Tape Drive)

The Read Drive Lines command allows the host computer to read the input lines from the target.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($11)
	$01	SCSI Command Flags	($00)
	$02 - $0B	Reserved.	

Request Length:

$00000000 - $00000001 (Bytes)

Transfer Length:

$00000000 - $00000001 (Bytes)

Buffer Data Structure:

Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

Errors:

Good
Check Condition.

$8012 - Inquiry; (All Devices)
[M]

This command retrieves information about the target device regarding various parameters and configuration of same.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($12)
	$01	Reserved	($00)
	$02	Reserved	($00)
	$03	VPD Identifier	($xx)
	$04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

The information returned by this call is displayed graphically below. Each field is defined as follows.

Byte 0 Peripheral Device Type.

00 = Direct-Access Device
01 = Sequential-Access Device
02 = Printer Device
03 = Processor Device
04 = Write-Once Read-Multiple Device
05 = Read-Only Direct-Access Device
06 = Scanner Device
07 = Optical Memory Device
08 = Changer Device
09 = Communications Device
0A - 0F = Reserved
10 = Direct Access Magnetic Tape Device
11 - 7E = Reserved
7F = Logical Unit not present
80 - FF = Vendor Unique

Byte 1, Bit 7 Removable Media Flag
Bit 0-6 Device Type Qualifier

Byte 2 **ANSI® Version**

0 Uses SCSI prior to x3.131-1986 approval
1 Current standard (ANSI® x3.131-1986).
2 - 7 Reserved

Byte 3 **Reserved**

Byte 4 **Additional Sense Length**

Byte \ Bit	7	6	5	4	3	2	1	0
0	Peripheral Device Type							
1	RMB	Device-Type Qualifier						
2	ISO Version		ECMA Version			ANSI-Approved Ver		
3	Reserved							
4	Additional Length							
	Vendor Unique Parameters							
5	Vendor Unique							
6-7	Reserved							
8-F	Vendor ID in ASCII							
10-1F	Product ID in ASCII							
20-23	Revision Level in ASCII							
24-25	Number of Resrve/Release extents							
26	Group Number							
27-2A	Group Commands Implemented							
2B	Group Number							
2C-2F	Group Commands Implemented							
n	$FF end of commands Implimented							

Errors:

Good
Check Condition.

NOTE: For addition information describing the required data returned by this call, please refer to the 'Apple SCSI Hard Disk Common Command Protocol' document. The information contained therein should apply to all devices that are to function in the Apple environment.

$8013 - Read QIC-100 Information; (Apple Tape Drive)

The Read QIC-100 Information command allows the host computer to read QIC-100 related information from the target.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($13)
	$01	SCSI Command Flags	($00)
	$02 - $0B	Reserved.	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

Errors:

Good
Check Condition.

$8014 - Recover Buffered Data; (Sequential Access)
[O]

This call is similar in function to the Read Command. The only difference being that the data requested is read from the target devices buffer memory rather than the media. This call is used to recover data from a terminated write call that was unable to write the data to the media from the buffer. This situation could arise during an asynchronous write call.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($14)
	$01	Reserved	(%00000000)
		Fixed	(%0000000x)
	$02 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $00xxxxxx (Bytes)
See FIXED bit definition in ANSI® and Vendor documents, driver may need to translate to Blocks)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

None.

Errors:

Good
Check Condition

$8014 - Recover Buffered Data; (Printer Devices)
[O]

This call is similar in function to the Read Command. The only difference being that the data requested is read from the target devices buffer memory rather than the media. This call is used to recover data from a terminated Print call that was unable to print the data to the media from the buffer. This situation could arise during an asynchronous print call.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($14)
	$01	Reserved	(%00000000)
	$02 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $00ffffff (Bytes)

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

None.

Errors:

Good
Check Condition

$8019 - Read QIC-100 Defect Data; (Apple Tape Drive)

The Read QIC-100 Defect Data command allows the host computer to read defect data from the controller. If this command follows a Format or Verify Unit command, the data returned to the host are the defects found for that verify. If the defect data is not in memory at the time this command is received, the manufacturers' block is read

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($19)
	$01	Immed	(%x0000000)
		CompLst	(%0000x000)
	$02 - $0B	Reserved.	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

Errors:

Good
Check Condition.

$801A - Mode Sense; (All Devices)
[O]

This command is complimentary to the Mode Select Command. It is used to get the Medium, Logical Unit, or Peripheral Device parameters from the target device.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($1A)
	$01	Page Format (PF)	(%000x0000)
		DBD Flag	(%0000x000)
	$02	Page Control	(%xx000000)
		Page Code	(%00xxxxxx)
	$03 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

The information returned by this call is displayed graphically below. Each field is defined as follows.

Byte 0	Sense Data Length
Byte 1	Medium Type
Byte 2, Bit 7	Write Protected flag
Byte 3	Block Descriptor Length

This is then followed by the Block Descriptor(s). Each block descriptor contains a block count and block length. Following this is any vendor unique information that the target may attach.

Byte \ Bit	7	6	5	4	3	2	1	0
0	Sense Data Length							
1	Medium Type							
2	WP	Reserved						
3	Block Descriptor Length							
	Block Descriptors							
0	Density Code							
1	Number of Blocks (MSB)							
2	Number of Blocks							
3	Number of Blocks (LSB)							
4	Reserved							
5	Block Length (MSB)							
6	Block Length							
7	Block Length (LSB)							
	Vendor Unique Parameters							
0 - n	Vendor Unique Parameter Byte(s)							

Errors:

Good

$801C - Receive Diagnostic Results; (All Devices) [O]

This command requests that the target device send it's analysis data after completing a Send Diagnostic Command. The data passed by this is vendor unique.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($1C)
	$01	Page Format (PF)	(%000x0000)
	$02 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

$00000000 - $0000ffff (Bytes)

Transfer Length:

$00000000 - $0000ffff (Bytes)

Buffer Data Structure:

Entirely Vendor Unique

Errors:

Good
Check Condition

$801F - Read Log; (Sequential Access Devices)
[O] See $805F command

The Read Log command is issued to sequential access devices to request statistical information maintained by the device. This is an optional command and the data is returned in the same format as the Mode Select /Sense page codes.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($1F)
	$01	Page Format (PF)	(%000x0000)
		No Log Save (NLS)	(%000000x0)
		No Log Clear (NLC)	(%0000000x)
	$02	Page Code	(%00xxxxxx)
	$03 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

The data is returned as vendor unique as well as Device or Medium specific

Errors:

Good
Check Condition

$8025 - Read Capacity; (Direct Access)
[M] (WORM Devices)
(Read-Only Direct Access)

This mandatory group 2 command causes the target device to return information about the capacity of the logical unit. This command also allows the application to determine the free space left before the next mechanical delay takes place. The flags used in this command are described in the vendor and ANSI® SCSI Specs.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($25)
	$01	RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB »» LSB)
	$06 - $07	Reserved	
	$08	PMI	(%0000000x)
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000008 All other values will be unpredictable or will return an error.

Transfer Length:

$00000008

Buffer Data Structure:

Eight bytes are returned. The first four are the Logical block address for the last block on the device if the PMI bit is zero. If the PMI bit is 1, then this is the last block before a significant delay will take place before the next block can be read. The last four bytes is the length of the block in bytes.

Bit / Byte	7	6	5	4	3	2	1	0
0	Logical Block Address (MSB)							
1	Logical Block Address							
2	Logical Block Address							
3	Logical Block Address (LSB)							
4	Block Length (MSB)							
5	Block Length							
6	Block Length							
7	Block Length (LSB)							

Errors:

Good

$8025 - Get Window Parameters; (Scanners)
[O]

The Get Window Parameters is used to pass information from the scanner detailing the task and how it is to be performed to the host. Before the scanner can scan a document or image, the application must provide certain details about the scan area. This information is provided in the form of parameters defining a scan window. These parameters include size, position, scanning resolution, scanning composition, as well as other parameters for each window.

The window parameters data shall consist of one or more Window Descriptor Blocks. Each window descriptor block specifies the location of a rectangular region and the mode in which the region is to be scanned.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($24)
	$01	Single	(%0000000x)
	$02 -$04	Reserved	
	$05	Window Identifier	($xx)
	$06 -$08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $00ffffff (Length of window descriptor in bytes)

Transfer Length:

$00000000 - $00ffffff (Length of transferred data in bytes)

Buffer Data Structure:

Below is the structure for the parameters to be transferred by this call. These are given here because of the lack of available documentation detailing this call. The descriptions below are extracted from several documents to give an over all comprehensive description of the parameters listed.

The Request Length above is the length, in bytes, of a window descriptor set. This means that the Window Descriptor Length times the number of descriptors sent should be eight less than the Request Length. In other words Request Length = Window Descriptor Length * number of descriptor blocks + 8.

All other bytes in the Descriptor header (the first 8 bytes) are reserved.

Bit / Byte	7	6	5	4	3	2	1	0
0 - 5	Reserved							
6	Window Descriptor Block Length (MSB)							
7	Window Descriptor Block Length (LSB)							
	Window Desriptor Block(s)							
0	Window Identifier							
1	Reserved							Auto
2-3	X Resolution (MSB»»LSB)							
4-5	Y Resolution (MSB»»LSB)							
6-9	Upper Left X (MSB»»LSB)							
A-D	Upper Left Y (MSB»»LSB)							
E-11	Width (MSB»»LSB)							
12-15	Length (MSB»»LSB)							
16	Brightness							
17	Threshold							
18	Contrast							
19	Image Composition							
1A	Bits per Pixel							
1B-1C	Halftone Patern (MSB»»LSB)							
1D	RIF	Reserved				Padding Type		
1E-1F	Bit Ordering (MSB»»LSB)							
20	Compression Type							
21	Compression Argument							
22-N	Reserved							

Each window descriptor contains information about one window.

The **Window Identifier** field contains a number between 0 and 255, which uniquely identifies the window defined by the block descriptor. Use this unique identifier to indicate each window during data transfers and status requests.

The **X Resolution** specifies the horizontal resolution in pixels per inch. A value of zero indicates that the scanner should use it's default resolution.

The **Y Resolution** specifies the vertical resolution in lines per inch. A value of zero indicates that the scanner should use it's default resolution.

The **Upper Left X** specifies the location of the X-coordinate of the upper left corner of this rectangular window and is measured in pixels as defined by X Resolution. The point 0,0 is considered the most upper-left corner of the window.

The **Upper Left Y** specifies the location of the Y-coordinate of the upper left corner of this rectangular window and is measured in lines as defined by Y Resolution. The point 0,0 is considered the most upper-left corner of the window.

The **Width** specifies the window width in pixels from left to right.

The **Length** specifies the window width in lines from top to bottom.

The **Brightness** has a range of one (lowest setting) through 255 (highest setting) with zero specifying the default value.

The **Threshold** is just that, the threshold setting of the scanner. A zero indicates that the scanner should use it's default setting for this. One is the lowest and 255 is the highest with 128 being the nominal setting.

The **Contrast** has a range of one (lowest setting) through 255 (highest setting) with zero specifying the default value.

The **Image** specifies the type of image acquired and is defined by the following table.

Code	Description
00	Bi-level black and white
01	Dithered/halftone black and white
02	Multi-level black and white (gray scale)
03	Bi-level RGB Color
04	Dithered/halftone RGB Color
05	Multi-level RGB Color
06 - FF	Reserved

The **Bits per Pixel** specifies the number of bits to be used to define each pixel. The higher the Image setting, the greater the number of bits required for each pixel.

The **Halftone Pattern** specifies the halftone process by which multi-level data is converted to binary data. This field shall be used in conjunction with the Image Composition code specified above.

The **Reverse Image Format** (RIF) bit is applicable only for images represented by one bit per pixel. A RIF bit of zero indicates that white pixels are to be indicated by zeros and black pixels are to be indicated by ones. A RIF bit of one indicates the opposite.

The **Padding Type** specifies what operation is to be done if the scanned data to be transmitted to the host is not an integral number of bytes. The padding type is defined below.

Code	Description
00	No padding
01	Pad with 0's to byte boundary
02	Pad with 1's to byte boundary
03	Truncate to byte boundary
04 - FF	Reserved

The **Bit Ordering** field defines the order in which data is transferred to the host from the window. Ordering will include direction of pixels in a scan line, direction of scan lines within a window and data packing within a byte.

The **Compression** type and **Argument** fields specify the compression technique to be applied to the scanned data prior to transmission to the host and are defined below.

Code	Compression Type	Argument
00	No compression	Reserved
01	CCITT Group III, 1 dimensional	Reserved
02	CCITT Group III, 2 dimensional	K factor
03	CCITT Group IV, 2 dimensional	Reserved
04 - 0F	Reserved	Reserved
10	Optical Character Recognition (OCR)	Vendor Unique
11 - 7F	Reserved	Reserved
80 - FF	Vendor Unique	Vendor Unique

Errors:

Good
Check Condition

$8028 - Read (Extended); (Direct Access Devices)
[M]

The Read (Extended) Command allows more than 256 blocks of data to be read in from the target device. It also allows for larger media. All other facets are the same as the $8008 Read Command. Please refer to the ANSI® Spec for details on the use of the flags.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($28)
	$01	DPO	(%000x0000)
		FUA	(%0000x000)
		RelAddr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB »» LSB)
	$06 - $08	Reserved.	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

Data

Errors:

Good
Check Condition
Reservation Conflict

$8028 - Read (Extended); (Scanner Devices)
[M]

The Read (Extended) Command allows more than 256 blocks of data to be read in from the target device. It also allows for larger media. All other facets are the same as the $8008 Read Command. Please refer to the ANSI® Spec for details on the use of the flags.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($28)
	$01	RelAddr	(%0000000x)
	$02	Transfer Data Type	($xx)
	$03	Reserved	
	$04 - $05	Transfer Identification	(MSB »» LSB)
	$06 - $08	Reserved.	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

Data

Errors:

Good
Check Condition
Reservation Conflict

$802D - Read Update Block; (Optical Memory)
[O]

This command requests that the target device transfer to the host data from the specified generation(s) and logical block(s). This command is more thoroughly discussed in the SCSI-2 ANSI® spec.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($2D)
	$01	DPO	(%000x0000)
		FUA	(%0000x000)
		RUBD	(%000000x0)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06	Latest	(%x0000000)
		Generation Addr (MSB)	(%0xxxxxxx)
	$07	Generation Addr (LSB)	
	$08	Reserved.	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

Data

Errors:

Good
Check Condition
Reservation Conflict

$8034 - Read Position; (Sequential Access Devices)
[O]

Status Command $8034 has two functions. The first function, Read Position, is an optional command for sequential access devices. It causes the target to return the current position of data blocks in the buffer and the position of the medium. When the buffer does not contain a whole block of data, or is empty, the two values are equal. No medium movement will result from this call.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($34)
	$01	Block Addr Type (BT)	(%0000000x)
	$02 - $08	Reserved.	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000014

Transfer Length:

$00000014

Buffer Data Structure:

The following text comes from the draft spec ANSI® x3.131-198x and all references to partitions relate to the SCSI concept of partitions and not those as implemented by Apple Computer, Inc.

Bit / Byte	7	6	5	4	3	2	1	0
0	BOP	LEP	Reserved			BPU	Reserved	
1	Partition Number							
2	Reserved							
3	Reserved							
4	Product Specific First Block Location (MSB)							
5	Product Specific First Block Location							
6	Product Specific First Block Location							
7	Product Specific First Block Location (LSB)							
8	Product Specific Last Block Location (MSB)							
9	Product Specific Last Block Location							
A	Product Specific Last Block Location							
B	Product Specific Last Block Location (LSB)							
C	Reserved							
D	Number of Blocks in Buffer (MSB)							
E	Number of Blocks in Buffer							
F	Number of Blocks in Buffer (LSB)							
10	Number of Bytes in Buffer (MSB)							
11	Number of Bytes in Buffer							
12	Number of Bytes in Buffer							
13	Number of Bytes in Buffer (LSB)							

Beginning of partition (BOP) bit, when set to one, indicates that the device's current logical position is at the beginning-of-partition. When set to zero it indicates that the current logical position is not at the beginning-of-partition. The BOP indication is not necessarily a result of a physical tape marker (e.g. reflective marker). When the partition number field is zero BOP, set to one, indicates that the position is at beginning-of-tape.

Logical end of partition (LEP) bit, when set to one, indicates that the device's current logical position is between early-warning and physical end-of-partition within the current partition. When set to zero it indicates that the current logical position is not in this area of the current partition. The LEP indication is not necessarily a result of a physical tape marker (e.g. reflective marker).

Block position unknown (BPU) bit, when set to one, indicates that the device's logical position is not known or cannot be obtained.

The partition number field reports the partition number for the current logical position.

Product specific first block location bytes indicate the relative position of the first data block within a partition. The value shall be the position of the next data block to be transferred between the initiator and the target if the previous command was a READ or a WRITE. The value shall be the position of the last block position transferred to the initiator if the previous command was a READ REVERSE. The information is product specific and is provided to be used with the LOCATE command to place the device's logical position at the appropriate logical block on another medium in the case of unrecoverable errors on the first medium.

Product specific last block location bytes indicate the relative position of the last data block within a partition. The value shall be the position of the next data block to be transferred between the buffer and the medium if the previous command was a READ or a WRITE. The value shall be the position of the last data block read from the medium into the target's buffer if the previous command was a READ REVERSE. The information is product specific and is provided to be used with the LOCATE command to place the device's logical position at the appropriate logical block on another medium in the case of unrecoverable errors on the first medium.

Number of blocks in buffer field indicates the number of data blocks in the target's buffer that have not been transferred to the medium.

Number of bytes in buffer field indicates the total number of data bytes in the target's buffer that have not been transferred.

Errors:

Good

$8034 - Get Data Status; (Scanner Devices)
[O]

Status Command $8034 has two functions. The second, Get Data Status, function is an optional command for scanner devices. It causes the target to return the scan data availability. The application program reports data availability for those windows that have been specified in the SCAN command. The format of the Get Data Status Command is shown below.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($34)
	$01	Wait	(%0000000x)
	$02 - $0B	Reserved.	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$02 - $0B	Reserved.	

The WAIT Flag indicates when the scanner will return data status to the host computer. A value of 1 indicates that the scanner is expected to wait for the quantity of data in the scanner's internal memory to exceed the limit set in the Mode Select command before responding with data. A value of 0 indicates that the target shall respond immediately with data whether or not data is available.

Request Length:

$00000000 - $0000ffff (Bytes)

Transfer Length:

$00000000 - $0000ffff (Bytes)

Buffer Data Structure:

The Get Data Status command returns the following structure providing a format for retrieving information about the availability of image data.

Bit / Byte	7	6	5	4	3	2	1	0
0	Data Transfer Length (MSB)							
1	Data Transfer Length							
2	Data Transfer Length (LSB)							
3	Reserved							
4	Window Identifier							
5	Reserved							
6	Buffer Space Available (MSB)							
7	Buffer Space Available							
8	Buffer Space Available (LSB)							
9	Scan Data Available (MSB)							
A	Scan Data Available							
B	Scan Data Available (LSB)							

The Data Length specifies the length in bytes of the data status available to be transferred to the host computer during the Get Data Status command. The data length size does not include the data length field. The data transferred to the host computer consists of multiple structures , each one comprising of 8 bytes each as defined by bytes $4 - $B in the table above. As defined by the window identifier, each returned structure is associated with a window descriptor.

The Block bit specifies the buffering capabilities of the scanner. A value of 1 indicates that the scanner must transfer all available data to the host computer before the scanner can generate more scan data. The bit is also set when the scanner has reached the end of the scan, even though the buffer may not be full. A value of 0 indicates that the scanner is not currently blocked due to a lack of available buffer space.

The Window Identifier parameter identifies the window associated with the returned structure. This value matches the window identifier in the Window Descriptor of the Define Window Parameters command. After completing the scan, the scanner indicates the end of the transfer by setting the data length to 0 and sends on data status.

The Buffer Space Available field indicates the amount of storage in bytes available for data transfers to the target. This field is valid only in scanners with the ability to accept data from a host computer for processing.

The Scan Data Available field indicates the amount of data in bytes available to be transferred from the target.

Errors:

Good

$8037 - Read Defect Data; (Direct Access Devices)
[O]

The Read Defect Data command requests that the target transfer the media defect data to the host computer. This command is optional for Direct Access Devices and is discussed in detail in the ANSI® x3.131-198x Spec.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($37)
	$01	Reserved	($00)
	$02	PList	(%000x0000)
		GList	(%0000x000)
		Defect List Format	(%00000xxx)
	$03 - $08	Reserved.	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $0000ffff (Bytes)

Transfer Length:

$00000000 - $0000ffff (Bytes)

Buffer Data Structure:

Please refer to the vendor and ANSI® SCSI documents for further details of this call.

Errors:

Good
Check Condition

$8038 - Read Element Status; (Changer Devices) [O]

The Read Element Status command will cause the target to report the status of it's internal elements. Refer to the Device and ANSI® documents for further details of this command and the flags associated with it.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($38)
	$01	**Reserved**	
	$02 - $03	Starting Element Address	(MSB»»LSB)
	$04 - $05	Number of Elements	(MSB»»LSB)
	$06 - $08	**Reserved**	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	**Reserved.**	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

Please refer to the vendor and ANSI® documents for details of the returned Element Status data.

Errors:

Check Condition

$803C - Read Buffer; (All Devices)
[O]

The Read Buffer command is used along with the Write Buffer command as a diagnostic tool for testing target memory and the SCSI Bus integrity. This command shall not alter the media. The reader should refer to the ANSI® x3.131-198x spec for more detailed information about this call.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($3C)
	$01	Mode	(%00000xxx)

Mode is defined as follows.

000	Combined Header and Data	Optional
001	Vendor Unique	Vendor Unique
010	Data	Optional
011	Descriptor	Optional
1xx	Reserved	Reserved

$02	Buffer ID	
$03 - $05	Buffer Offset	(MSB »» LSB)
$06 - $08	Reserved.	
$09	Vendor Unique	(%xx000000)
	Reserved	(%00xxxxxx)
$0A - $0B	Reserved.	

Request Length:

$00000000 - $00ffffff (Bytes)

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

See ANSI® and vendor documentation for details pertaining to this command

Errors:

Good
Check Condition

$803E - Read Long; [O] (Direct Access Devices)

The Read Long command request that the target transfer data to the host computer. This data is implementation specific, but shall include the data bytes and the ECC bytes. Any other data correctable by ECC should also be included.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($3E)
	$01	CORRECT	(%000000x0)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB »» LSB)
	$06 - $08	Reserved.	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $0000ffff (Bytes)

Transfer Length:

$00000000 - $0000ffff (Bytes)

Buffer Data Structure:

Data from the target including ECC and other information (Implementation specific).

Errors:

Good
Check Condition

$805A - Mode Sense; (All Devices)
[O]

This command is complimentary to the Mode Select Command. It is used to get the Medium, Logical Unit, or Peripheral Device parameters from the target device.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($5A)
	$01	Page Format (PF)	(%000x0000)
	$02	Page Control	(%xx000000)
		Page Code	(%00xxxxxx)
	$03 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $0000ffff (Bytes)

Transfer Length:

$00000000 - $0000ffff (Bytes)

Buffer Data Structure:

Refer to vendor and ANSI® document for further details and data descriptions.

Errors:

Good

$805F - Read Log; (**Sequential Access Devices**)
[O] See **$801F command**

The Read Log command is issued to sequential access devices to request statistical information maintained by the device. This is an optional command and the data is returned in the same format as the Mode Select/Sense page codes.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($5F)
	$01	Page Format (PF)	(%000x0000)
		No Log Save (NLS)	(%000000x0)
		No Log Clear (NLC)	(%0000000x)
	$02	Page Code	(%00xxxxxx)
	$03 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $0000ffff (Bytes)

Transfer Length:

$00000000 - $0000ffff (Bytes)

Buffer Data Structure:

The data is returned as vendor unique as well as Device or Medium specific

Errors:

Good
Check Condition

$80A8 - Read; (Optical Media)
[M]

This call is mostly a duplication of the $8028 command except for larger transfer lengths. Please refer to the $8028 command documentation.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($A8)
	$01	DPO	(%000x0000)
		FUA	(%0000x000)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06 - $0A	Reserved.	
	$0B	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

Data

Errors:

Good
Check Condition
Reservation Conflict.

$80AD - Read Update Block; (Optical Media)
[O]

This call is mostly a duplication of the $802D command except for larger transfer lengths. Please refer to the $802D command for field definitions.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($AD)
	$01	DPO	(%000x0000)
		FUA	(%0000x000)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06	Latest	(%x0000000)
		Generation Addr (MSB)	(%0xxxxxxx)
	$07	Generation Addr (LSB)	
	$08 - $0A	Reserved.	
	$0B	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

Data

Errors:

Good
Check Condition
Reservation Conflict

$80B7 - Read Defect Data; (Optical Media)
[O]

This call is mostly a duplication of the $8037 command except for larger transfer lengths. The ANSI® spec that defines this call is currently incomplete. Until more information is made available this call will be allocated but remain undefined and unsupported.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($B7)
	$01	PList	(%000x0000)
		GList	(%0000x000)
		Defect List Format	(%00000xxx)
	$02 - $0A	Reserved.	
	$0B	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)

Request Length:

$00000000 - $ffffffff (Bytes)

Transfer Length:

$00000000 - $ffffffff (Bytes)

Buffer Data Structure:

Defect Data. This contains an eight byte header followed by zero or more defect descriptors. See ANSI® and vendor documentation for details of this data.

Errors:

Good
Check Condition

$80C1 - Read TOC; (Ruby Drive)

This command requests that the target device transfer the the TOC (Table of Contents) to the host computer. Details of this call are given in the 'Apple CD ROM SCSI Command Set Rev. 1.2'.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($C1)
	$01	SCSI Command Flags	($00)
	$02	Track Number	(TNO in BCD)
	$03 - $04	Reserved	
	$05	TOC Type	(%xx000000)
	$06 - $0B	Reserved.	

Request Length:

$00000000 - $0000ffff (Bytes)

Transfer Length:

$00000000 - $0000ffff (Bytes)

Buffer Data Structure:

See the 'Apple CD ROM SCSI Command Set Rev. 1.2'

Errors:

Good
Check Condition

$80C2 - Read Q Subcode; (Ruby Drive)

This call causes the target device to send the Q Subcode data to the host computer. Details of this call are given in the 'Apple CD ROM SCSI Command Set Rev. 1.2'.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($C2)
	$01	SCSI Command Flags	($00)
	$02 - $0B	Reserved.	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

See the 'Apple CD ROM SCSI Command Set Rev. 1.2'

Errors:

Good
Check Condition

$80C3 - Read Header; (Ruby Drive)

This to the host computer four bytes of header information for the specified logical block address. This call is similar to the Read Extended ($8028) command, but only the header bytes are returned by the target. Details of this call are given in the 'Apple CD ROM SCSI Command Set Rev. 1.2'.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($C3)
	$01	SCSI Command Flags	($00)
	$02 - $05	Logical Block	(MSB »» LSB)
	$06 - $0B	Reserved.	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

See the 'Apple CD ROM SCSI Command Set Rev. 1.2'

Errors:

Good
Check Condition

$80CC - Audio Status; (Ruby Drive)

Use of this command causes the target device to transmit the current audio play status and the starting Q Subcode Address of the next track. Details of this call are given in the 'Apple CD ROM SCSI Command Set Rev. 1.2'.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($CC)
	$01	SCSI Command Flags	($00)
	$02 - $0B	Reserved.	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

See the 'Apple CD ROM SCSI Command Set Rev. 1.2'

Errors:

Good
Check Condition

Control Call

Call Parameters :	Device Number	≠	\$0000
	Call Number	=	\$0006
	Control List Pointer		
	Request Count		
	Transfer Count		
	Control Code		
	DIB Pointer		

Device Number: *This word parameter specifies the target device. This parameter must be nonzero.*

Call Number: *This word parameter specifies the type of call.*

Control List Ptr: *This longword points to the control list location.*

Request Count: *This longword parameter indicates the number of bytes to be transferred. If the request count is smaller than the minimum buffer size required by the call, an error is returned.*

Transfer Count: *This longword returned by the call indicates the number of bytes actually transferred.*

Control Code: *This word parameter specifies the type of control request. Control codes of \$0000 through \$7FFF are standard control calls that must be supported by device drivers. SCSI specific control calls use control codes in the range of \$8000 through \$FFFF. A list of standard control calls follows:*

\$0000	Reset Device
\$0001	Format Device
\$0002	Eject
\$0003	Set Configuration Parameters
\$0004	Set Wait/No Wait Mode
\$0005	Set Format Options
\$0006	Assign Partition Owner
\$0007	Arm Event
\$0008	Disarm Event
\$0009	Set Partition Map
\$000A - \$7FFF	Reserved - These control codes to be assigned by Apple Computer, Inc.

The list of device specific status calls are as follows:

\$8000 - \$80FF Device specific SCSI commands.

The following are the SCSI specific calls. The values to the right are the Devices that use that code and are defined at the end of the list.

Group 0 Commands

$8001	ReZero	1,5,6,8,B,E
	Rewind Unit	2
	Reset Printer	D
$8002	Down Load Code	D
$8004	Format Unit (Non-Printers)	1,8
$8004	Format Unit (Printers)	3,D,E
$8005	Draw Bits	D
	Send QIC-100 System Data	E
$8006	Clear Bits	D
$8007	Reassign Blocks	1,5,8,E
$8009	Verify Unit	E
$800A	Write	1,2,5,8,E
	Print	3,D
	Send	4
	Send Message	A
$800B	Seek	1,5,6,8,B,E
	Track Select	2
	Slew and Print	3
$800C	Reserved	
$800F	Write Controller Information	E
$8010	Write File Marks	2
	Flush Buffer	3
	Drive Pass-Thru	E
$8011	Space	2
$8013	Verify	2
$8014	Write QIC-100 Information	E
$8015	Mode Select	1,2,3,5,6,7,8,A,B,C,D,E
$8016	Reserve Unit	1,2,3,5,6,7,8,B,C,D,E
$8017	Release Unit	1,2,3,5,6,7,8,B,C,D,E
$8018	Copy	0
$8019	Erase	2
$801B	Start/Stop Unit	1,5,6,8,B
	Load/Unload	2,E
	Stop Print	3
$801B	Scan	7,C
$801D	Send Diagnostic	0,A,B,C,E
$801E	Prevent/Allow Medium Removal	1,2,5,6,8,B

Group 1 Commands

$8020 - $8023	Reserved	
$8024	Define Window Parameters	7,C
$8026 - $8027	Reserved	
$802A	Write	1,5,8,E
	Send	7,C
$802B	Seek	1,5,6,8,B,E
	Locate	2
$802C	Erase	8
	???????Read Generation????????	8
$802E	Write and Verify	1,5,8
$802F	Verify	1,5,6,8,B
$8030	Search Data High (Unsupported)	1,5,6,8
$8031	Search Data Equal (Unsupported)	1,5,6,8
	Medium Position	7
$8032	Search Data Low (Unsupported)	1,5,6,8
$8033	Set Limits	1,5,6,8
$8034	Pre_Fetch	1,8
$8035	Synchronize Cache	1,8
$8036	Lock/Unlock Cache	1,8
$8038	Media Scan	8
$8039	Compare	1,2,5,6,7,8
$803A	Copy and Verify	1,2,5,6,7,8
$803B	Write Buffer	1,2,7,8,A,B,E
$803D	Update Block	8
$803F	Write Long	1

Group 2 Commands

$8040	Change Definition(Unsupported)	0
$8041 - $8054	Reserved	
$8055	Mode Select	0
$8056 - $805F	Reserved	

Group 3 Commands

$8060 - $807F	Reserved

Group 4 Commands

$8080 - $809F	Reserved

Group 5 Commands

\$80A0 - \$80A4	Reserved	
\$80A5	Move Medium	9
\$80A6	Exchange Medium	9
\$80A7	Reserved	
\$80A9	Reserved	
\$80AA	Write	8
\$80AB	Reserved	
\$80AC	Erase	8
\$80AE	Write and Verify	8
\$80AF	Verify	8
\$80B0	Search Data High (Unsupported)	8
\$80B1	Search Data Equal (Unsupported)	8
\$80B2	Search Data Low (Unsupported)	8
\$80B3	Set Limits	8
\$80B4 - \$80B6	Reserved	
\$80B8 - \$80BC	Reserved	
\$80BD	Update Blocks	8
\$80BE - \$80BF	Reserved	

Group 6 Commands

\$80C0	Eject Disk	B
\$80C4 - \$80C7	Reserved	
\$80C8	Audio Track Search	B
\$80C9	Audio Play	B
\$80CA	Audio Pause	B
\$80CB	Audio Stop	B
\$80CD	Audio Scan	B
\$80CE - \$80DF	Reserved	

Group 7 Commands

$80E0 - $80FF Reserved

Device Definitions

0	=	Devices '1' through '9'
1	=	Direct-Access Devices
2	=	Sequential-Access Devices
3	=	Printer Devices
4	=	Processor Devices
5	=	Write-Once Read-Multiple Devices
6	=	Read-Only Direct-Access Devices
7	=	Scanner Devices
8	=	Optical Memory Devices
9	=	Changer Devices
A	=	Communications Devices
B	=	Apple CD_ROM Drive
C	=	Apple SCSI Scanner
D	=	Apple LaserWriter SC
E	=	Apple Tape Drive

Non-SCSI Commands

$8100 - $FFFF Reserved

DIB Pointer: *This longword points to the DIB for the target device.*

This call is used to send control information to the device or device itself and may be used to issue any extended SCSI Command that sends data to the device through the use of device specific control codes. The device driver is responsible for validating the status code prior to executing the requested command. The following 65816 code sample shows how this might be done.

```
validate_ctrl_code    lda   control_code          ;Get control Code
                      bmi   device_specific       ;Is it Device Specific?
                      cmp   #max_command+1        ;No. Is it out of range?
                      blt   do_standard           ;No. Goto code segment
bad_code_exit         lda   #BAD_CODE             ;Central BAD CODE Exit
                      sec
                      rtl

device_specific       and   #$00ff                ;Is it ≤ the max SCSI
                      pha                         ;command for this
                      inc                         ;device?
                      ldy   #SCSI_maxcmd_offset
                      cmp   [DIB_PTR],y
                      blt   so_far_so_good        ;It's in a good range
                      pla                         ;Restore stack
                      bra   bad_code_exit         ;exit with error
```

```
so_far_so_good      lda     1,s                     ;Get control code from stack
                    lsr                             ;Generate a group index
                    lsr                             ;to use to get to the
                    lsr                             ;correct group offset
                    lsr
                    and     #$000e
                    tay
                    lda     group_offset,y
                    sta     temp                    ;Save offset for later

                    lda     1,s                     ;Get control code from stack
                    and     #$0010                  ;Upper or lower half
                    beq     first_16_bits           ;of group bitmap?
                    inc     temp                    ;Upper half. Adjust
                    inc     temp                    ;offset by two

first_16_bits       pla                             ;Last time. Get control code
                    and     #$000f                  ;Retain low nibble
                    asl     a                       ;Account for 16 bit
                    tay                             ;table of offsets.
                    lda     cmd_bm_mask,y
                    ldy     temp
                    and     [DIB_PTR],y             ;Is bit set in bitmap?
                    beq     bad_code_exit           ;No. Error exit.
                            .
                            .
                            .
max_command         equ     $0009                   ;Max standard control code

SCSI_maxcmd_offset  equ     $0042

group_offset        dc      i2'GROUP0-DIB_start'    ;Offsets from start
                    dc      i2'GROUP1-DIB_start'    ;of DIB to Group
                    dc      i2'GROUP2-DIB_start'    ;bitmaps
                    dc      i2'GROUP3-DIB_start'
                    dc      i2'GROUP4-DIB_start'
                    dc      i2'GROUP5-DIB_start'
                    dc      i2'GROUP6-DIB_start'
                    dc      i2'GROUP7-DIB_start'

temp                ds      2

cmd_bm_mask         dc      i2'%1000000000000000'   ;Bitmap masks.
                    dc      i2'%0100000000000000'
                    dc      i2'%0010000000000000'
                    dc      i2'%0001000000000000'
                    dc      i2'%0000100000000000'
                    dc      i2'%0000010000000000'
                    dc      i2'%0000001000000000'
                    dc      i2'%0000000100000000'
                    dc      i2'%0000000010000000'
                    dc      i2'%0000000001000000'
                    dc      i2'%0000000000100000'
                    dc      i2'%0000000000010000'
                    dc      i2'%0000000000001000'
                    dc      i2'%0000000000000100'
                    dc      i2'%0000000000000010'
                    dc      i2'%0000000000000001'
```

If an invalid control code is passed to the driver, the driver will return a 'BAD CODE' error. The driver may also wish to identify where the call came from and shield certain calls from the application. Based on this, the driver could use a secondary command bitmap to validate codes allowed from the application. If an invalid control list length is passed to the driver, the driver should return a 'BAD PARAMETER' error. The device driver sets the transfer count to the number of bytes processed as a result of the control call.

NOTE: Both standard and device specific status calls may detect an OFFLINE or DISKSWITCH status. If either of these conditions are detected then a **SET_DISKSW** call is issued to set the device dispatcher maintained disk switched error.

$0000 - Reset Device

This control call is used to reset a particular device to it's default settings. A device driver should configure itself based on the contents of the driver's configuration parameter list (not to be confused with control list). This call should also return any media variables modified through a Set_Format_Options call to the default settings.

Control List: Word Length of control list ($0000)

$0001 - Format Device

This control call is used to format the media used by a block device. This call is not linked to any particular file system. It simply prepares all blocks on the media for reading and writing. After completion of formatting a block device, any media variables that may have been modified by a Set_Format_Options control call should be returned to their default value. Character devices do not support this function and will return with no error.

Due to Partitioning, this command will check the device for a valid partition map. If none exists, then a format call will be sent to the device. If on the other hand a partition map does exist then this call will result in no error with no action taken on the device. If a hard format is desired, then an $8004 Format Unit command should be used to force the Format command being issued.

Control List: Word Length of control list ($0000)

$0002 - Eject

This control call is used to physically or logically eject the media from a block device. The targeted device will be sent an SCSI Unload/Unlock command to enable the ejection of the media and then physically ejected if the device supports the eject call. If not, then the device will be marked offline and it will be up to the user to then remove the media. If the device is linked to other devices, it will be marked offline and physical ejection will occur when all the linked DIBs for that device are marked offline. Character devices that do not support this control call, will return with no error.

Control List:	Word	Length of control list ($0000)

$0003 - Set Configuration Parameters

This control call is used to send device specific configuration parameters to a device. The first word in the control list indicates the length of the configuration parameter list in bytes. The configuration parameters should be placed into the configuration list contiguous to the byte count. The structure of the configuration parameter list is device dependent. The first word of the new configuration list must be equal to the request count. Additionally, the first word of the new configuration list must equal the first word of the existing configuration list. It is not legal to set a new configuration list of a different size of the existing list. If this call is made with an erroneous configuration list length a bad parameter error will be returned.

Control List:	Word	Length of configuration parameter list
	Data	Configuration Parameter List Data

$0004 - Wait/No Wait Mode

This call is used to set a character device to Wait or No Wait mode. If in wait mode, the device will wait for the number of characters specified in the request count of a read call before returning from the read. If in No Wait mode, a read call will return immediately with a transfer count indicating the number of characters returned. If a character was available the transfer count will return from a read with a non zero value. If a character was not available the transfer count will return from a read call with a value of zero. The control list will contain a word with a value of $0000 to set Wait Mode or a value of $8000 to set No Wait mode. Block devices do not support this control call and will return with no error.

Control List:	Word	Length of control list ($0002)
	Word	Wait/No Wait Mode

$0005 - Set Format Options

This call sets media specific parameters prior to executing a format call to a block device. This call does not imply a format. The control list consists of a word (Format_RefNum) and a word (Interleave_Factor). The format reference number specifies a group of variables used during a subsequent format call which includes Format Environment, Block Count, Block Size and Interleave Factor. If the Interleave_Factor is set to NIL then the default interleave specified in the format variables list is used. In order to obtain a list of Format_RefNum values and their corresponding variables, a Get_Format_Options status call is issued to the device. After the appropriate format variables are selected, a Set_Format_Options control call is issued followed by a format control call. This call is not supported by character devices and should return a 'BAD_COMMAND' error.

Control List:	Word	Format_RefNum
	Word	Interleave_Factor if used.

NOTE: **This call must be issued to the first DIB in the link if the device is partitioned. If this is not the case an Invalid Device Number error will be returned.**

$0006 - Assign Partition Owner

This call is supported by block devices supporting partitioned media. This call is executed by an FST as a result of the 'ERASE_DISK' system call. The control list consists of a class 1 string indicating the partition type. Partition types can be up to 32 bytes in length. If the string is less than 32 bytes in length, it must be terminated with a NUL. A partition type can be cleared by setting the first byte of the string to the NUL character. Upper and lower case characters are considered equivalent. The driver will then reassign the current partition to the new owner. This call does not reassign physical block allocation within a device partition descriptor. Block devices utilizing non-partitioned media and character devices should return with no error.

Control List:	String	Class 1 string specifying partition type

Example String
$0d $00 $41 $70 $70 $6c $65 $5f $50 $52 $4f $44 $4f $53 $00

Example Types
Apple_MFS
Apple_HFS
Apple_Unix_SVR2
Apple_partition_map
Apple_Driver
Apple_PRODOS
Apple_Free
Apple_Scratch

$0007 - Arm Signal

This call provides a means for a device driver to install a signal handler into the GS/OS signal manager's handler list. Signal code is an arbitrary value assigned by the driver to identify the signals that it generates. The driver will maintain a list of armed signals. When an interrupt associated with the driver occurs, the driver's interrupt handler will assess the priority of the interrupt and pass the appropriate signal handler's address to the signal mechanism. Each signal should have a unique signal code. If an attempt is made to arm a signal for which that signal code already exists in the drivers signal list, a driver_bad_parameter error will be returned and the signal will not be added to the drivers signal handler list. Priority is the signal priority, with $0000 being the lowest priority and $FFFF being the highest priority. Handler address is the address where the signal handler resides.

Control List:	Word	Signal Code
	Word	Priority
	Longword	Signal Handler Address

NOTE: **This call is not supported by the drivers that follow the layout defined by this spec. The Application should use async and data chaining coding techniques to fully benefit from the designs speed enhancements.**

$0008 - Disarm Signal

This call provides a means for a device driver to remove it's interrupt handler into the GS/OS signal manager's handler list. Signal code is an arbitrary value assigned by the driver when the signal was armed.

Control List:	Word	Signal Code

NOTE: **This call is not supported by the drivers that follow the layout defined by this spec. The Application should use async and data chaining coding techniques to fully benefit from the designs speed enhancements.**

$0009 - Set Partition Map

This call writes the partition map to the device specified which must be the first device of any linked list. If not an **Invalid Device Number** error is returned.

The structure of the partition map entries are described in detail in 'Inside Macintosh™ Volume V' under the SCSI Manager section and is shown below. It must be noted that all fields, except strings, in the partition map are stored High Byte »» Low Byte. Example: Even though the partition signature is defined as $504d, it will appear to the 65xxx family of processors as $4d50.

byte	Field	Description
0	pmSig (word)	always $504d
2	pmSigPad (word)	reserved for future use
4	pmMapBlkCnt (long word)	number of blocks in map
8	pmPyPartStart (long word)	first physical block of partition
C	pmPartBlkCnt (long word)	number of blocks in partition
10	pmPartName (32 bytes)	partition name
30	pmPartType (32 bytes)	partition type
50	pmLgDataStart (long word)	first logical block of data area
54	pmDataCnt (long word)	number of blocks in data are
58	pmPartStatus (long word)	partition status information
5C	pmLgBootStart (long word)	first logical block of boot code
60	pmBootSize (long word)	size in bytes of boot code
64	pmBootLoad (long word)	boot code load address
68	pmBootLoad2 (long word)	additional boot load information
6C	pmBootEntry (long word)	boot code entry point
70	pmBootEntry2 (long word)	additional boot entry information
74	pmBootCksum (long word)	boot code checksum
78	pmProcessor (16 bytes)	processor type
88	(128 bytes)	boot specific arguments

The size of the partition map can vary in size[3] but must be written all at once. The driver validates the request count against the pmMapBlkCnt using the block size for that device. If pmMapBlkCnt * block size ≠ Request Count then an **Invalid Byte Count** error is returned. If the application wishes to reassign an entry to a different type, the $0006 control call Assign Partition Owner should be used.

When the driver receives this call it must do several layers of verification before it can be considered successfully completed. First the driver attempts to ensure that there is enough space to allocate DIBs for these partitions.

Example: The device had four partitions, each with it's own DIB. The new map has nine entries. The driver must allocate memory for five additional DIBs. If this can not be accomplished then a **Resource Not Available** error will be returned. Next the driver issues a **Post Driver Install** to see if the device table has room for the additional devices. If not, the memory allocated will be released and the appropriate error will be returned. Finally, the driver will write the partition map to the device.

The driver treats the successful completion of this call as if the media is removable and will declare to the device that a disk switched condition exists. No error will be **returned.**

---

3 The partition map can contain as few as 2 (Partition map plus the partition itself) or as many as 32 entries maximum.

Device Specific Control Calls

The following calls are the device specific control calls allowed by this driver. For a discussion on how to issue these calls, please refer to the Device Specific Status calls.

$8001 - ReZero Unit; (Direct Access Devices)
[O]

The ReZero Unit command instructs the target device to place itself in a specific state. This is an optional command to several of the block device classes. Refer to the vendor specifications for details.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($01)
	$01 - $03	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$8001 - Rewind Unit; (Sequential Access Devices)
[M]

This is the equivalent to the $8001 ReZero Unit call only this is mandatory to Sequential access devices. The command requests that the target rewind the logical unit to the beginning-of-medium or load point. Prior to execution of this command, the target shall write any data currently in the buffer to the media.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($01)
	$01	Immed	(%0000000x)
		If set to one, the status will be returned as soon as the operation has been initiated. Zero indicates that the status is returned after the operation is complete.	
	Note:	Prior to issuing a Rewind command with the Immed bit set to one, it is suggested that the Write Filemarks command with the Immed bit set to zero be used to flush the devices buffer to the media.	
	$02 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$8002 - Down Load Code; (Apple LaserWriter SC)

This command is specific to the LaserWriter SC and provides a method for the host computer to download 68000 code to the LaserWriter SC in order to correct errors in the firmware or to expand it's functionality.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($02)
	$01 - $0B	Reserved.	

Request Length:

$00000000 - $00ffffff (Bytes)

This specifies the number of bytes to send. If the LaserWriter SC lacks sufficient available memory, the command is terminated with a Check Condition status, the ILI bit of the Sense Data set to one, and the Information Byte of the Sense Data set to the difference between the request length and the amount of available memory. The Request Length must be an EVEN number (the 68000 requires word alignment). The last word of the data sent must be a two's compliment checksum of all the previous words of data. If the checksum doesn't match, a Check Condition status is returned and the Sense Key set to Aborted Command. After the code is downloaded and checksum verified, execution begins at the first byte of the code.

NOTE: The download command is meant for advanced programmers with knowledge of the internals of the LaserWriter SC firmware.

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

Code.

Errors:

Good
Check Condition

$8004 - Format Unit; (Direct Access Devices)
[M]

The format command instructs the target device to format the currently mounted media so that all host addressable logical blocks can be accessed. The media may also be certified at this time and various levels of defects reported. There is no guarantee that the media has or has not been altered. For further details pertaining to the levels of defect management, please refer to the vendor specification as well as the ANSI® x3.131-1986 document.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($04)
	$01	FmtData (format data) bit	(%000x0000)
		CmpLst (complete list) bit	(%0000x000)
		Defect List Format	(%00000xxx)
	$02	Vendor Unique	
	$03 - $04	Interleave	(MSB»»LSB)
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

See spec for defect data types and structures.

Errors:

Good
Check Condition

$8004 - Format Unit; (Printer devices)

The format command to printers provides a means for the host computer to specify forms or fonts to printers that support programmable forms or fonts. The format information sent is vendor unique since it is peripheral-device specific, For further details please refer to the vendor specification as well as the ANSI® x3.131-1986 document.

NOTE: **The LaserWriter SC printer does not follow the ANSI® Spec for this call. It is important that the programmer refer to the spec for this device. If not used correctly, this command can cause imaging to occur off the paper and will reduce the life of the Canon Engine.**

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($04)
	$01	FmtData (format data) bit	(%000000xx)
		00 = Set Form	
		01 = Set Font	
		10 = Vendor Unique	
		11 = Reserved	
	$02 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

$00000000 - $00ffffff (Bytes)

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

See vendor spec for the details of this command.

Errors:

Good
Check Condition

$8005 - Send QIC-100 System Data; (Apple Tape Drive)

The Send QIC-100 System Data command allows the host computer to send 128 bytes of QIC-100 System Data to the controller. Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($05)
	$01 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

$00000000 - $00000080 (Bytes)

Transfer Length:

$00000000 - $00000080 (Bytes)

Buffer Data Structure:

Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

Errors:

Good
Check Condition.

$8005 - Draw Bits; (Apple LaserWriter SC)

This device specific command applies to only one device, the LaserWriter SC. This command transfers image data into the specified rectangular area of imaging memory using one of several transfer modes. This command is only appropriate for devices with at least one megabyte of RAM installed.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($05)
	$01 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

$00000000 - $00ffffff (Bytes)

NOTE: The LaserWriter SC spec states that this value must be $A (10). The driver hill handle this transparent to the caller. The length of the entire structure being sent must be correctly represented here.

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

See vendor spec for the details of this command.

Errors:

Good
Check Condition

$8006 - Clear Bits; (Apple LaserWriter SC)

This is the compliment to the $8005 Draw Bits command and it applies to only one device, the LaserWriter SC. This command clears image data from the specified rectangular area of imaging memory. This command is only appropriate for devices with at least one megabyte of RAM installed.

The **Command Data** structure is defined as:

Byte $00		SCSI Command Number	($06)
$01 - $04		Reserved.	
$05		Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
$06 - $0B		Reserved.	

Request Length:

$00000008 (Bytes)

Transfer Length:

$00000008 (Bytes)

Buffer Data Structure:

All pixels within the bounds rectangle only will be cleared. If the bounds rectangle has it's bottom right point higher or more left than it's top left point, the rectangle is assumed to contain no bits and no image data will be cleared. A bounds rectangle outside of the imaging rectangle (Set by the format command) is an error.

Bit / Byte	7	6	5	4	3	2	1	0
0	Bounds.topLeft.x (MSB)							
1	Bounds.topLeft.x (LSB)							
2	Bounds.topLeft.y (MSB)							
3	Bounds.topLeft.y (LSB)							
4	Bounds.botRight.x (MSB)							
5	Bounds.botRight.x (LSB)							
6	Bounds.botRight.y (MSB)							
7	Bounds.botRight.y (LSB)							

Errors:

Good
Check Condition

$8007 - Reassign Blocks; (Direct Access Devices)
[O] (WORM Devices)

The Reassign Blocks command requests the target to reassign the defective blocks listed in the defect list to an area of the logical unit set aside for this purpose.

A defect list is transferred to the target with this command containing the logical blocks to be reassigned.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($07)
	$01 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

$00000000 - $0000ffff (Bytes)

This value is to be equal to the number of defective blocks in the list times 4 plus 4. If eight blocks ar to be reassigned, then the request length will be 8*4+4 or 36 ($24)

Transfer Length:

$00000000 - $0000ffff (Bytes)

Buffer Data Structure:

The defect descriptor specifies a four byte defect logical block address that contains the defect. The defect descriptors are in ascending order.

If the target has insufficient capacity to reassign all of the defective logical blocks, the command shall terminate with a Check Condition status and the sense key shall be set to Medium Error. The logical block address of the first logical block not reassigned is returned in the information bytes of the sense data.

Bit / Byte	7	6	5	4	3	2	1	0
0	Reserved							
1	Reserved							
2	Reserved							
3	Reserved							
4	Defect Logical Block Address (MSB)							
5	Defect Logical Block Address							
6	Defect Logical Block Address							
7	Defect Logical Block Address (LSB)							
n-3	Defect Logical Block Address (MSB)							
n-2	Defect Logical Block Address							
n-1	Defect Logical Block Address							
n	Defect Logical Block Address (LSB)							

Errors:

Good
Check Condition

$8009 - Verify Unit; (Apple Tape Drive)

The Verify Unit command causes the controller to perform a verify operation of the format of the tape. This is done by reading every frame on tape and determining whether both CRCs of that frame can be computed correctly. Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($09)
	$01	SCSI Command Flags	(%x0000000)
		Defect List Format	(%000xxxxx)
	$02	OneTrk	(%x0000000)
		MfgBlk	(%0x000000)
		ConCar	(%00x00000)
		Track Number	(%000xxxxx)
	$03 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

Errors:

Good
Check Condition.

$800A - Write; (Block Devices)
[M]

This is the standard SCSI Write Command used by most devices that receive data from the caller. If a block address greater than $ffff is needed, a command $802A should be used.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($0A)
	$01	Reserved	
	$02 - $03	Logical Block Address	(MSB »» LSB)
	$04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

None

Errors:

Good
Check Condition
Reservation Conflict

NOTE: See also **$800A - Write, Print, Send and Send Message** described below.

$800A - Write; (Sequential Devices)
[M]

This is the standard SCSI Write Command used by most devices that receive data from the caller. If a byte count greater than $ffffff is needed, a command $802A should be used.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($0A)
	$01	Fixed	(%0000000x)
	$02 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $00ffffff (Bytes)

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

Data

Errors:

Good
Check Condition
Reservation Conflict

NOTE: See also **$800A - Write** above and **Print, Send** and **Send Message** described below.

$800A - Print; [M] (Printer Device)

This command sends the specified number of bytes to the target device for printing.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($0A)
	$01 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $00ffffff (Bytes)

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

Data to be sent or printed.

Errors:

Good
Check Condition

NOTE: The LaserWriter SC uses the Print command in a slightly different manner. When issued, no data is sent with the command. The LaserWriter SC will print the data that has been sent via the Draw Bits and Clear Bits command along with other data that resides in it's RAM.

NOTE: See also $800A - Write and Write above and Send and Send Message described below.

$800A - Send; (Processor Devices)
[M]

This command sends the requested number of bytes to the target Processor Device.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($0A)
	$01	Async Event (AEN)	(%0000000x)
	$02 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $00ffffff (Bytes)

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

Data to be sent or printed.

Errors:

Good
Check Condition

NOTE: See also **$800A - Write, Write and Print** above and **Send Message** described below.

$800A - Send Message; (Communication Devices)
[M]

This command transfers the requested number of bytes to the target communication device.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($0A)
	$01 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $00ffffff (Bytes)

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

Data to be sent or printed.

Errors:

Good
Check Condition

NOTE: See also **$800A - Write, Write, Print,** and **Send** above.

$800B - Seek; **(Block Devices)**
[O]

This command requests that the target position itself so as to be in position to read or write the Logical block specified. If a block address greater than $ffff is required, then use the $802B Seek (Extended) command.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($0B)
	$01	Reserved	
	$02 - $03	Logical Block Address	(MSB »» LSB)
	$04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

NOTE: See also **$800B - Track Select and Slew and Print** described below.

$800B - Track Select; **(Sequential Access Devices)**
[O]

The Track Select command instructs the target device to to select the track specified by the Track Value. This is an optional command.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($0B)
	$01 - $03	Reserved	
	$04	Track Value	($xx)
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

NOTE: See also **$800B - Seek** above and **Slew and Print** described and below.

$800B - Slew and Print; (Printers)
[O]

The Slew and Print command transfers the requested number of bytes to the target device to be printed. The data sent is application dependent. This command is provided for printers that do not support forms control information imbedded within the print data.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($0B)
	$01	Channel	(%0000000x)
	$02	Slew Value	
	$03 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

If the Channel bit is zero, the slew value specifies the number of lines that the form shall advance before printing the data. A slew value of $FF that the form is advanced to the first line of the next form. If the Channel bit is one, the Slew Value indicates the forms control channel number to be advanced to.

Request Length:

$00000000 - $0000ffff (Bytes)

Transfer Length:

$00000000 - $0000ffff (Bytes)

Buffer Data Structure:

Data to be printed.

Errors:

Good
Check Condition

NOTE: See also **$800B - Seek and Track** Select described above.

$800F - Write Controller Information; (Apple Tape Drive)

The Write Controller Information command allows the host to input. Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($0F)
	$01 - $0B	Reserved.	

Request Length:

$00000006

Transfer Length:

$00000006

Buffer Data Structure:

Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

Errors:

Good
Check Condition.

$8010 - Write File Marks; (Sequential Access Devices)
[M]

This command causes the target device to write out the requested number of file marks to the media at the current position. By specifying zero filemarks it is possible to force the device to flush any data that might still be buffered to the media. See the ANSI® Spec for details about this call.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($10)
	$01	Immed	(%0000000x)
	$02 - $04	Number of file marks	(MSB»»LSB)
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

NOTE: See also **$8010 - Flush Buffer** described below.

$8010 - Flush Buffer; (Printers)
[O]

This command instructs the printer to finish printing any data that remains in it's buffers. This is useful for applications that may wish to ensure that certain housekeeping chores have been completed. See ANSI® Document for more details. This is an optional command for printer devices.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($10)
	$01 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

NOTE: See also $8010 - **Write Filemarks** described above.

$8010 - Drive Pass-Thru; (Apple Tape Drive)

The Drive Pass-Thru command allows the host to directly access the drive level interface. Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($10)
	$01	Reserved	
	$02	Register 1	
	$03	Register 2	
	$04	Reserved	
	$05	CmdMode	(%x0000000)
	$06 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

Errors:

Good
Check Condition.

$8011 - Space; (Sequential Access)
[M]

The Space command is currently an optional command for Sequential Access Devices that is headed for the mandatory class under SCSI-2 currently in draft. This command provides several positioning functions that are determined by the code and count. Both forward and backward positioning are possible, although some target devices may not support all of the combinations. These targets will return a Check Condition status and an Illegal Request sense key for any illegal request. Please refer to the ANSI® x3.131-1986 spec for further details.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($11)
	$01	Code	(%000000xx)
		00 = Blocks	
		01 = Filemarks	
		10 = Sequential Filemarks	
		11 = Logical End-of-Data	
	$02 - $04	Count	(MSB»»LSB)

This count specifies the number of blocks or filemarks to skip over. If the number is positive, it will result in forward movement of the target media. A negative (2's compliment) value will cause the media to moved in the reverse direction over count blocks or filemarks.

$05	Vendor Unique	(%xx000000)
	Reserved	(%00xxxxxx)
$06 - $0B	Reserved	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$8013 - Verify; (Sequential Access)
[M]

The Verify command cause the target to verify one or more blocks from the current position. There are two flags associated with this call. For further information about this call and it's implementation please refer to the ANSI® x3.131-1986 spec.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($13)
	$01	Immed	(%00000x00)
		BytCmp	(%000000x0)
		Fixed	(%0000000x)
	$02 - $04	Verification Length	(MSB»»LSB)
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$8014 - Write QIC-100 Information; (Apple Tape Drive)

The Write QIC-100 Information command allows the host to write QIC-100 information to the controller, which will then be written on tape upon execution of the next Write or Format commands. Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($14)
	$01 - $0B	Reserved.	

Request Length:

$0000000b

Transfer Length:

$0000000b

Buffer Data Structure:

Please refer to the MCD-40/SCSI & DM Reference Manual for the details of this call.

Errors:

Good
Check Condition.

$8015 - Mode Select; (**Direct Access Devices**)
[O] (**Sequential Access**)
(**Printer Devices**)
(**Scanner Devices**)

The Mode Select command allows the host system to specify various parameters (Medium, Logical Unit, or Peripheral device) to the target. The targets that implement this command must also implement the Mode Sense command. This call applies to many of the SCSI Device Types and the developer should refer to the device specifications as well as the correct section of the ANSI® x3.131-1986 document for further details.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($15)
	$01	Page Format (PF)	(%000x0000)
		Save Pages (SP)	(%0000000x)
	$02 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

The structure varies depending on the targeted device type. Please see the device documentation as well as the ANSI® spec for byte by byte details of the Mode Select Parameter List.

Errors:

Good
Check Condition

$8015 - Mode Select; (Changer Devices)
[O]

The Mode Select command allows the host system to specify various parameters (Medium, Logical Unit, or Peripheral device) to the target. The targets that implement this command must also implement the Mode Sense command. This call applies to many of the SCSI Device Types and the developer should refer to the device specifications as well as the correct section of the ANSI® x3.131-1986 document for further details.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($15)
	$01	Save Pages (SP)	(%0000000x)
	$02 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

The structure varies depending on the targeted device type. Please see the device documentation as well as the ANSI® spec for byte by byte details of the Mode Select Parameter List.

Errors:

Good
Check Condition

$8015 - Mode Select; (Communications Devices)
[O]

The Mode Select command allows the host system to specify various parameters (Medium, Logical Unit, or Peripheral device) to the target. The targets that implement this command must also implement the Mode Sense command. This call applies to many of the SCSI Device Types and the developer should refer to the device specifications as well as the correct section of the ANSI® x3.131-1986 document for further details.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($15)
	$01 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

The structure varies depending on the targeted device type. Please see the device documentation as well as the ANSI® spec for byte by byte details of the Mode Select Parameter List.

Errors:

Good
Check Condition

$8016 - Reserve Unit; (**Direct Access Devices**)
[M] (**WORM Devices**)
(**Read-Only Direct Access**)

The Reserve Unit command is used to reserve a unit on the SCSI Bus for a particular host system and is used mostly in multi-host configurations. This command is supported to give the application full access to the target devices on the system. The reader should refer to the device and ANSI® Documentation to determine if this command offers a function that they need.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($16)
	$01	3rdPty	(%000x0000)
		3rd Party Device ID	(%0000xxx0)
		Extent	(%0000000x)
	$02	Reservation Ident	
	$03 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $0000ffff (Bytes)

Transfer Length:

$00000000 - $0000ffff (Bytes)

Buffer Data Structure:

Refer to documentation.

Errors:

Good
Check Condition
Reservation Conflict

NOTE: See also $8016 - Reserve Units described below.

$8016 - Reserve Unit; (Sequential Access Devices)
[M] (Printer Devices)
(Scanner Devices)

The Reserve Unit command is used to reserve a unit on the SCSI Bus for a particular host system and is used mostly in multi-host configurations. This command is supported to give the application full access to the target devices on the system. The reader should refer to the device and ANSI® Documentation to determine if this command offers a function that they need.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($16)
	$01	3rdPty	(%000x0000)
		3rd Party Device ID	(%0000xxx0)
	$02 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

Refer to documentation.

Errors:

Good
Check Condition
Reservation Conflict

NOTE: See also **$8016 - Reserve Units** described above and below.

$8016 - Reserve Unit; (Apple Tape Drive)
[M]

The Reserve Unit command is used to reserve a unit on the SCSI Bus for a particular host system and is used mostly in multi-host configurations. This command is supported to give the application full access to the target devices on the system. The reader should refer to the 3M MCD-40 DM/SCSI Device as well as the ANSI® Documentation to determine if this command offers a function that they need.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($16)
	$01	3rdPty	(%000x0000)
		3rd Party Device ID	(%0000xxx0)
		Extent	(%0000000x)
	$02	Reservation Identification	
	$03 - $04	Reserved	
	$05	Reserved	(%00000000)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $0000xxxx (Bytes)
In multiples of 8

Transfer Length:

$00000000 - $0000xxxx (Bytes)
In multiples of 8

Buffer Data Structure:

Refer to documentation.

Errors:

Good
Check Condition
Reservation Conflict

NOTE: See also $8016 - Reserve Units described above and below.

$8016 - Reserve Unit; (Changer Devices)
[M]

The Reserve Unit command is used to reserve a unit on the SCSI Bus for a particular host system and is used mostly in multi-host configurations. This command is supported to give the application full access to the target devices on the system. The reader should refer to the device and ANSI® Documentation to determine if this command offers a function that they need.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($16)
	$01	3rdPty	(%000x0000)
		3rd Party Device ID	(%0000xxx0)
		Element	(%0000000x)
	$02	Reservation Ident	
	$03 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

$00000000 - $0000ffff (Bytes)

Transfer Length:

$00000000 - $0000ffff (Bytes)

Buffer Data Structure:

Refer to documentation.

Errors:

Good
Check Condition
Reservation Conflict

NOTE: See also **$8016 - Reserve Units** described above.

$8017 - Release Unit; (Direct Access Devices)
[M] (WORM Devices)
(Read-Only Direct Access)

This is the compliment command to Reserve Unit. This instructs the target to release the specified, previously reserved area.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($17)
	$01	3rdPty	(%000x0000)
		3rd Party Device ID	(%0000xxx0)
		Extent	(%0000000x)
	$02	Reservation Ident	
	$03 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

NOTE: See also $8017 - Release Units described below.

$8017 - Release Unit; (Sequential Access Devices)
[M] (Printer Devices)
(Scanner Devices)

This is the compliment command to Reserve Unit. This instructs the target to release the specified, previously reserved area.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($17)
	$01	3rdPty	(%000x0000)
		3rd Party Device ID	(%0000xxx0)
	$02 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

NOTE: See also $8017 - Reserve Units described above and below.

$8017 - Release Unit; (Changer Devices)
[M]

This is the compliment command to Reserve Unit. This instructs the target to release the specified, previously reserved area.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($17)
	$01	3rdPty	(%000x0000)
		3rd Party Device ID	(%0000xxx0)
		Element	(%0000000x)
	$02	Reservation Ident	
	$03 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

NOTE: See also $8017 - Reserve Units described above.

$8019 - Erase; (**Sequential Access Devices**)
[M]

This command instructs the target device erase part or all of the remaining media beginning at the current media position. 'Erased' means either the media shall be erased or a pattern shall be written that appears as a gap to the target device. The distance is controlled by the long bit. A long bit of one indicates that all remaining media on the target device shall be erased. A long bit of zero indicates that a peripheral device specified portion of the media only will be erased.

Some targets may reject the Erase command with the long bit set to one if the media is not at the beginning of the media.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($19)
	$01	Immed	(%000000x0)
		Long	(%0000000x)
	$02 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$801B - Start/Stop Unit; [O] (Direct Access Devices) (WORM Devices) (Read-Only Direct Access)

The Start/Stop command instructs the target to enable or disable the device for further operations. An Immed bit of one requests that the status be returned as soon as the operation is initiated. An Immed bit of zero indicates that the target is to wait until the request is completed before sending the status code.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($1B)
	$01	Immed	(%0000000x)
	$02 - $03	Reserved	
	$04	Load/Eject	(%000000x0)
		Start	(%0000000x)
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

NOTE: See also $801B - Load/Unload and Stop Print described below.

$801B - Load/Unload; [O] **(Sequential Access Devices)**

The Load/Unload command instructs the target to prepare, unload, or retension the media. The load operation is in addition to the autoload performed when the media was inserted or powered up. An Immed bit of one requests that the status be returned as soon as the operation is initiated. An Immed bit of zero indicates that the target is to wait until the request is completed before sending the status code.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($1B)
	$01	Immed	(%0000000x)
	$02 - $03	Reserved	
	$04	End of Tape (EOT)	(%00000x00)
		Retension Re-Ten	(%000000x0)
		Load	(%0000000x)
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

NOTE: See also **$801B - Stop Print** described below or **Start/Stop Unit** above.

$801B - Stop Print; (Printers)

The Stop Print command causes a buffered printer device to halt any printing in an orderly fashion.

A retain bit of zero indicates that any buffered data is to be discarded while a one indicates that the data is to be preserved. The Recover Buffered Data command would be used to recover the data for a subsequent Print command, or a Slew and Print can be used to cause the remaining unrecovered data to be printed followed by any new data sent by the command. Where or at which point the printing is suspended is unspecified.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($1B)
	$01	Retain Bit	(%0000000x)
	$02	Vendor Unique	
	$03 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

NOTE: See also $801B - Load/Unload described below or Start/Stop Unit above.

$801B - Scan; **(Scanner Devices)**
[O]

The Scan Command is used to inform the target that we are ready to receive data from it and how much data we have room for or have requested.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($1B)
	$01 - $04	Reserved	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

The buffer contains one or more window identifier bytes. The identifier tells the target which window descriptor to use for this command. These are the same that are defined by the Define Window Parameters call.

Errors:

Good

$801D - Send Diagnostic; (All Devices)
[M]

The Send Diagnostic command instructs the target to perform diagnostic test on itself, or the attached peripheral devices, or both. This command is usually followed by a Receive Diagnostic Results command except when doing a self test.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($1D)
	$01	Page Format (PF)	(%000x0000)
		SelfTest	(%00000x00)
		DevOfL	(%000000x0)
		UnitOfL	(%0000000x)
	$02 - $04	Reserved.	
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

$00000000 - $0000ffff (Bytes diagnostic list)

Transfer Length:

$00000000 - $0000ffff (Bytes diagnostic list sent)

Buffer Data Structure:

The diagnostic parameter list is device specific. Refer to the Device manuals for further information about this structure.

Errors:

Good
Check Condition

$801E - Prevent/Allow Removal; (Direct Access)
[O] (WORM Devices)
(Read-Only Direct Access)
(Sequential Access)

The Prevent/Allow Medium Removal command instructs the target to enable or disable the removal of the media from the unit.

A prevent bit of one inhibits mechanisms that normally allow removal of the media. A zero allows the media to be removed. The prevent condition shall be terminated after a Prevent/Allow Medium Removal command with the prevent bit set to zero, or Bus Device Reset condition from the host or a 'hard' Reset condition.

Targets that contain cache memory shall logically perform a Flush Cache command for the entire media prior to allowing media removal.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($1E)
	$01 - $03	Reserved	
	$04	Prevent Flag	(%0000000x)
	$05	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$06 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$8024 - Define Window Parameters; (Scanners)
[M]

The Define Window Parameters is a mandatory command for all scanners. It is used to pass information to the scanner detailing the task and how it is to be performed. Before the scanner can scan a document or image, the application must provide certain details about the scan area. This information is provided in the form of parameters defining a scan window. These parameters include size, position, scanning resolution, scanning composition, as well as other parameters for each window.

If sent, the window parameters data shall consist of one or more Window Descriptor Blocks. Each window descriptor block specifies the location of a rectangular region and the mode in which the region is to be scanned.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($24)
	$01 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $00ffffff (Length of window descriptor in bytes)

Transfer Length:

$00000000 - $00ffffff (Length of transferred data in bytes)

Buffer Data Structure:

Below is the structure for the parameters to be transferred by this call. These are given here because of the lack of available documentation detailing this call. The descriptions below are extracted from several documents to give an over all comprehensive description of the parameters listed.

The Request Length above is the length, in bytes, of a window descriptor. This means that the Window Descriptor Length times the number of descriptors sent should be eight less than the Request Length. In other words Request Length = Window Descriptor Length * number of descriptor blocks + 8.

All other bytes in the Descriptor header (the first 8 bytes) are reserved.

Byte \ Bit	7	6	5	4	3	2	1	0
0 - 5	Reserved							
6	Window Descriptor Block Length (MSB)							
7	Window Descriptor Block Length (LSB)							
	Window Desriptor Block(s)							
0	Window Identifier							
1	Reserved							Auto
2-3	X Resolution (MSB»»LSB)							
4-5	Y Resolution (MSB»»LSB)							
6-9	Upper Left X (MSB»»LSB)							
A-D	Upper Left Y (MSB»»LSB)							
E-11	Width (MSB»»LSB)							
12-15	Length (MSB»»LSB)							
16	Brightness							
17	Threshold							
18	Contrast							
19	Image Composition							
1A	Bits per Pixel							
1B-1C	Halftone Patern (MSB»»LSB)							
1D	RIF	Reserved				Padding Type		
1E-1F	Bit Ordering (MSB»»LSB)							
20	Compression Type							
21	Compression Argument							
22-N	Reserved							

Each window descriptor contains information about one window.

The **Window Identifier** field contains a number between 0 and 255, which uniquely identifies the window defined by the block descriptor. Use this unique identifier to indicate each window during data transfers and status requests.

The **X Resolution** specifies the horizontal resolution in pixels per inch. A value of zero indicates that the scanner should use it's default resolution.

The Y **Resolution** specifies the vertical resolution in lines per inch. A value of zero indicates that the scanner should use it's default resolution.

The **Upper Left X** specifies the location of the X-coordinate of the upper left corner of this rectangular window and is measured in pixels as defined by X Resolution. The point 0,0 is considered the most upper-left corner of the window.

The **Upper Left Y** specifies the location of the Y-coordinate of the upper left corner of this rectangular window and is measured in lines as defined by Y Resolution. The point 0,0 is considered the most upper-left corner of the window.

The **Width** specifies the window width in pixels from left to right.

The **Length** specifies the window width in lines from top to bottom.

The **Brightness** has a range of one (lowest setting) through 255 (highest setting) with zero specifying the default value.

The **Threshold** is just that, the threshold setting of the scanner. A zero indicates that the scanner should use it's default setting for this. One is the lowest and 255 is the highest with 128 being the nominal setting.

The **Contrast** has a range of one (lowest setting) through 255 (highest setting) with zero specifying the default value.

The **Image** specifies the type of image acquired and is defined by the following table.

Code	Description
00	Bi-level black and white
01	Dithered/halftone black and white
02	Multi-level black and white (gray scale)
03	Bi-level RGB Color
04	Dithered/halftone RGB Color
05	Multi-level RGB Color
06 - FF	Reserved

The **Bits per Pixel** specifies the number of bits to be used to define each pixel. The higher the Image setting, the greater the number of bits required for each pixel.

The **Halftone Pattern** specifies the halftone process by which multi-level data is converted to binary data. This field shall be used in conjunction with the Image Composition code specified above.

The **Reverse Image Format** (RIF) bit is applicable only for images represented by one bit per pixel. A RIF bit of zero indicates that white pixels are to be indicated by zeros and black pixels are to be indicated by ones. A RIF bit of one indicates the opposite.

The **Padding Type** specifies what operation is to be done if the scanned data to be transmitted to the host is not an integral number of bytes. The padding type is defined below.

Code	Description
00	No padding
01	Pad with 0's to byte boundary
02	Pad with 1's to byte boundary
03	Truncate to byte boundary
04 - FF	Reserved

The **Bit Ordering** field defines the order in which data is transferred to the host from the window. Ordering will include direction of pixels in a scan line, direction of scan lines within a window and data packing within a byte.

The **Compression type** and **Argument** fields specify the compression technique to be applied to the scanned data prior to transmission to the host and are defined below.

Code	Compression Type	Argument
00	No compression	Reserved
01	CCITT Group III, 1 dimensional	Reserved
02	CCITT Group III, 2 dimensional	K factor
03	CCITT Group IV, 2 dimensional	Reserved
04 - 0F	Reserved	Reserved
10	Optical Character Recognition (OCR)	Vendor Unique
11 - 7F	Reserved	Reserved
80 - FF	Vendor Unique	Vendor Unique

Errors:

Good
Check Condition

$802A - Write (Extended); (Direct Access Devices) [M]

The $802A Write command is similar to the $800A Write command only the Logical Block Address and the Transfer capabilities have been expanded. There are also some additional flags and the reader should refer to the device documentation for details about these flags and their function.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($2A)
	$01	DPO	(%000x0000)
		FUA	(%0000x000)
		Write Same (WrtSme)	(%00000x00)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB »» LSB)
	$06 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

Data

Errors:

Good
Check Condition
Reservation Conflict

NOTE: See also **$802A - Send and Write** described below.

$802A - Send (Extended); (Scanner Devices)
[O]

This command is similar in function to the write command except parameter not storage data is sent to the scanner.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($2A)
	$01	RelAdr	(%0000000x)
	$02	Transfer Data Type	
	$03	Reserved	
	$04 - $05	Transfer Identification	(MSB»»LSB)
	$06 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $00ffffff (Bytes)

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

Refer to the device documents for a description of this as well as the Transfer Data Type and Transfer Identification values.

Errors:

Good
Check Condition

NOTE: See also **$802A - Write** above and below.

$802A - Write (Extended); (Optical Memory Devices)
[O]

The $802A Write command is similar to the $800A Write command only the Logical Block Address and the Transfer capabilities have been expanded. There are also some additional flags and the reader should refer to the device documentation for details about these flags and their function.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($2A)
	$01	DPO	(%000x0000)
		FUA	(%0000x000)
		Erase By Pass (EBP)	(%00000x00)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB »» LSB)
	$06 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

Data

Errors:

Good
Check Condition
Reservation Conflict

NOTE: See also **$802A - Send and Write** described above.

$802B - Seek (Extended); (Direct Access Devices)
[O] (WORM Devices)
(Read-Only Direct Access)

This is an extended seek command for devices that support more block address than the standard seek allows.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($2B)
	$01	Reserved	
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

NOTE: See also **$802B - Locate** below.

$802B - Locate (Extended); (Sequential Access Devices)

The Locate command instructs the target device to position the media so that the block address specified is positioned for the next write call. The data written will be sent to this block address. If any data resides in the targets buffer, it will be written before the Locate command is executed. Refer to the Device and ANSI® documents for details concerning the flags and partition values for this command.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($2B)
	$01	Blk Address Type (BT)	(%00000x00)
		Change Partition (CP)	(%000000x0)
		Immed	(%0000000x)
	$02	Reserved	
	$03 - $06	Logical Block Address	(MSB»»LSB)
	$07	Reserved	
	$08	Partition	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

NOTE: See also $802B - Seek above.

$802C - Erase; (Optical Memory)
[O]

The Erase command instructs the target device to erase or fill with blank pattern part or all of the remaining media starting with the block specified. See device and ANSI® documents for flag definition and usage.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($2C)
	$01	Erase All (ERA)	(%00000x00)
		RelAdr	(%0000000x)
	$02 - $05	Starting Logical Blk Addr	(MSB»»LSB)
	$06	Reserved	
	$07 - $08	Number of Blocks	(MSB»»LSB)
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition
Reservation Conflict

NOTE: <u>This command seems to conflict with the $802C command **Read Generation**. The ANSI® spec is unclear. As the ANSI® document is updated, these will be modified to reflect those changes.</u>

$802C - Read Generation; (Optical Memory)
[O]

The Erase command instructs the target device to erase or fill with blank pattern part or all of the remaining media starting with the block specified. See device and ANSI® documents for flag definition and usage.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($2C)
	$01	RelAdr	(%0000000x)
	$02 - $05	Logical Blk Addr	(MSB»»LSB)
	$06 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

Generation data. See vendor and ANSI® documentation.

Errors:

Good

NOTE: This command seems to conflict with the $802C command **Erase**. The ANSI® spec is unclear. As the ANSI® document is updated, these will be modified to reflect those changes.

$802E - Write and Verify; (Direct Access Devices)
[O]

The Write and Verify command requests that the target device write the data send and then after the write, verify that what is on the media is what the host sent. Refer to the Device and ANSI® documents for information concerning the flags and their usage.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($2E)
	$01	DPO	(%000x0000)
		WrtSme	(%00000x00)
		BytChk	(%000000x0)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

Data to be written and verified

Errors:

Good
Check Condition
Reservation Conflict

NOTE: See also **$802E - Write and Verify** below.

$802E - Write and Verify; (Optical Memory Devices)
[O]

The Write and Verify command requests that the target device write the data send and then after the write, verify that what is on the media is what the host sent. Refer to the Device and ANSI® documents for information concerning the flags and their usage.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($2E)
	$01	DPO	(%000x0000)
		EBP	(%00000x00)
		BytChk	(%000000x0)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

Data to be written and verified

Errors:

Good
Check Condition
Reservation Conflict

NOTE: See also **$802E - Write and Verify** above.

$802F - Verify; (**Direct Access Devices**)
[O]

The Verify command is almost identical to the Write and Verify command except that no data is send or written. the target verifies the data that is on the media. Refer to the Device and ANSI® documents for information concerning the flags and their usage.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($2F)
	$01	DPO	(%000x0000)
		BytChk	(%000000x0)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06	Reserved	
	$07 - $08	Number of blocks	(MSB»»LSB)
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

None.

Transfer Length:

None.

Buffer Data Structure:

None.

Errors:

Good
Check Condition

NOTE: See also **$802F - Verify** below.

$802F - Verify; (Optical Memory Devices)
[O] (WORM Devices)

The Verify command is almost identical to the Write and Verify command except that no data is send or written. the target verifies the data that is on the media. Refer to the Device and ANSI® documents for information concerning the flags and their usage.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($2F)
	$01	DPO	(%000x0000)
		Blank Verify (BlkVfy)	(%00000x00)
		BytChk	(%000000x0)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06	Reserved	
	$07 - $08	Number of blocks	(MSB»»LSB)
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

None.

Transfer Length:

None.

Buffer Data Structure:

None. The request length specifies the number of blocks to be verified.

Errors:

Good
Check Condition

NOTE: See also $802F - Verify above.

$8031 - Medium Position; (Scanner Devices)
[O]

The Medium Position command provides a variety of media positioning functions. Absolute as well as relative positioning of the medium is provided, although some SCSI devices may only support a subset of this command. Such SCSI devices shall return a Check Condition Status. Please refer to the Device and ANSI® documents for further details concerning this command and is use.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($31)
	$01	Position Type	(%00000xxx)
	$02 - $04	Count	
	$05 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$8033 - Set Limits; [U] (Direct Access Devices) (WORM Devices) (Read-Only Direct Access)

The Set Limits command defines the range within which subsequent linked commands may operate. A second Set Limits command may not be linked to a chain of commands in which a Set Limits command has already been issued.

A read inhibit (RdInh) bit of one indicates that read operations within the range are inhibited. A write inhibit (WrInh) bit of one indicates that write operations within the range are inhibited.

The logical block address specifies the starting address for the range. The number of blocks specifies the number of blocks within the range. A number of zero indicates that the range shall extend to the last logical block on the logical unit.

Any attempt to access outside of the restricted range or any attempt to perform an inhibited operation within the restricted range shall not be performed.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($33)
	$01	Read Inhibit (RdInh)	(%000000x0)
		Write Inhibit (WrInh)	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06	Reserved	
	$07 - $08	Number of Blocks	(MSB»»LSB)
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$8034 - Pre Fetch; **(Direct Access Devices)**
[O]

The Pre Fetch command request the target to transfer the requested blocks to the targets cache memory. No data is transferred to the host computer.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($34)
	$01	Immed	(%000000x0)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06	Reserved	
	$07 - $08	Number of blocks	(MSB»»LSB)
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

None.

Transfer Length:

None.

Buffer Data Structure:

None

Errors:

Good
Condition Met
Check Condition

$8035 - Synchronize Cache; (Direct Access Devices)
[O]

The Synchronize Cache command ensures that all logical blocks within the specified range have their most recent data values recorded on the physical media. For each logical block within the specified range, if a more recent data value exists in the cache memory than on the physical media, then the logical block will be written from the cache to the media. Blocks are not necessarily removed from the cache by this command.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($35)
	$01	Immed	(%000000x0)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06	Reserved	
	$07 - $08	Number of Blocks	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$8036 - Lock/Unlock Cache; (Direct Access Devices)
[O]

The Lock/Unlock Cache command requests that the target disallow or allow logical blocks within the specified range to be removed from the cache memory by the target's cache replacement algorithm. Locked logical blocks may be written to the physical media when modified, but a copy of the modified logical block shall remain in the cache.

A lock bit of one indicates that any logical block in the specified range that is currently present in the cache shall be locked into the cache. Only logical blocks that are already present in the cache are actually locked. A lock bit of zero indicates that all logical blocks in the specified range that are currently locked into the cache shall be unlocked, but not necessarily removed.

The logical block address specifies the first logical block of the range to be locked. The number of blocks specifies the total number of contiguous blocks to be considered for locking or unlocking. A number of zero indicates that all the remaining blocks shall be considered.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($36)
	$01	Lock	(%000000x0)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06	Reserved	
	$07 - $08	Number of Blocks	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$8038 - Media Scan; (**Optical Memory Devices**)
[O] (**WORM Devices**)

The Media Scan command will cause the target to scan a defined range searching for a contiguous span of the media either empty or written. Results are posted in the Sense Data Block. Refer to the Device and ANSI® documents for further details of this command and the flags associated with it.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($38)
	$01	Writn Blk Search (WBS)	(%000x0000)
		Adv Scan Alg (ASA)	(%0000x000)
		Rev Search Dir (RSD)	(%00000x00)
		Part Rslt Accept (PRA)	(%000000x0)
		RelAdr	(%0000000x)
	$02 - $05	Beg. Logical Blk Address	(MSB»»LSB)
	$06 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 or $00000008 (Bytes)

Transfer Length:

$00000000 or $00000008 (Bytes)

Buffer Data Structure:

Number of Blocks to Scan and Number of Blocks to Verify are specified in the following optional parameter block. If this block is omitted by a Request Length of zero, the Number of Blocks to Scan shall default to zero (scan to end of media) and the Number of Blocks to Verify shall default

$00	Number of Blocks to Verify (MSB)
$01	Number of Blocks to Verify
$02	Number of Blocks to Verify
$03	Number of Blocks to Verify (LSB)
$04	Number of Blocks to Scan (MSB)
$05	Number of Blocks to Scan
$06	Number of Blocks to Scan
$07	Number of Blocks to Scan (LSB)

Errors:

Good
Check Condition
Condition Met

$803B - Write Buffer;

The Write Buffer command is used along with the Read Buffer command as a diagnostic tool for testing target memory and the SCSI Bus integrity. This command shall not alter the media. The reader should refer to the ANSI® x3.131-198x spec for more detailed information about this call.

The **Command Data** structure is defined as:

Byte	$00		SCSI Command Number	($3B)
	$01		Mode	(%00000xxx)
		000	Combined Header and Data	Optional
		001	Vendor Unique	Vendor Unique
		010	Data	Optional
		011	Descriptor	Optional
		100	Download Microcode	Optional
		101	Download Microcode and Save	Optional
		11x	Reserved	Reserved
	$02		Buffer ID	
	$03 - $05		Buffer Offset	(MSB »» LSB)
	$06 - $08		Reserved	
	$09		Vendor Unique	(%xx000000)
			Reserved	(%00xxxxxx)
	$0A - $0B		Reserved.	

Request Length:

$00000000 - $00ffffff (Bytes)

Transfer Length:

$00000000 - $00ffffff (Bytes)

Buffer Data Structure:

See vendor and ANSI® x3.131-198x documents for details pertaining to this command

Errors:

Good
Check Condition

$803D - Update Block; (**Optical Memory Devices**)
[O] (**WORM Devices**)

The Update Block command logically replaces data on the media with new data. Please refer to the vendors device spec and the ANSI® documents for details about the implementation of this command.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($3D)
	$01	RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

Data to be used in update.

Errors:

Good
Check Condition

$803F - Write Long; (Direct Access Devices)
[O]

The Write Long command request that the target transfer data from the host computer. This data is implementation specific, but shall include the data bytes and the ECC bytes. Any other data correctable by ECC should also be included.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($3F)
	$01	RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB »» LSB)
	$06 - $08	Reserved	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved.	

Request Length:

$00000000 - $0000ffff (Bytes)

Transfer Length:

$00000000 - $0000ffff (Bytes)

Buffer Data Structure:

Data to the target including ECC and other information (Implementation specific).

Errors:

Good
Check Condition

$8055 - Mode Select; (All Devices)
[O]

The Mode Select command allows the host system to specify various parameters (Medium, Logical Unit, or Peripheral device) to the target. The targets that implement this command must also implement the Mode Sense command. This call applies to many of the SCSI Device Types and the developer should refer to the device specifications as well as the correct section of the ANSI® document for further details.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($55)
	$01	Page Format (PF)	(%000x0000)
		Disable Blk Desript (DBD)	(%0000x000)
	$02	Page Control (PC)	(%xx000000)
		Page Code	(%00xxxxxx)
	$03 - $08	Reserved.	
	$09	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)
	$0A - $0B	Reserved	

Request Length:

$00000000 - $0000ffff (Bytes)

Transfer Length:

$00000000 - $0000ffff (Bytes)

Buffer Data Structure:

The structure varies depending on the targeted device type. Please see the device documentation as well as the ANSI® spec for byte by byte details of the Mode Select Parameter List.

Errors:

Good
Check Condition

$80A5 - Move Medium; (Changer Devices)
[M]

This command requests that the target device move a unit of media between two elements of the automatic media changer.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($A5)
	$01	Reserved	
	$02 - $03	Transport Element Addr	(MSB»»LSB)
	$04 - $05	Source Address	(MSB»»LSB)
	$06 - $07	Destination Address	(MSB»»LSB)
	$08 - $0A	Reserved.	
	$0B	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)

Request Length:

Unused.

Transfer Length:

Unused.

Buffer Data Structure:

None.

Errors:

Good
Check Condition

$80A6 - Exchange Medium; (Changer Devices) [O]

This call allows the host system to request the target device to exchange a piece of media from the transport element with one at a full element. Please refer to the vendor and ANSI® documents for details.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($A6)
	$01	Reserved	
	$02 - $03	Transport Element Addr	(MSB»»LSB)
	$04 - $05	Source Address	(MSB»»LSB)
	$06 - $07	First Destination Address	(MSB»»LSB)
	$08 - $09	Second Destination Addr	(MSB»»LSB)
	$0A	Inv1	(%000000x0)
		Inv2	(%0000000x)
	$0B	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)

Request Length:

Unused.

Transfer Length:

Unused.

Buffer Data Structure:

None.

Errors:

Good

$80AA - Write; (Optical Memory)
[M]

This call is mostly a duplication of the $802A command except for larger transfer lengths.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($AA)
	$01	DPO	(%000x0000)
		FUA	(%0000x000)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06 - $0A	Reserved.	
	$0B	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

Data

Errors:

Good

$80AC - Erase; **(Optical Memory)**
[O]

The Erase command instructs the target device to erase or fill with blank pattern part or all of the remaining media starting with the block specified. See device and ANSI® documents for flag definition and usage.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($AC)
	$01	Erase All (ERA)	(%00000x00)
		RelAdr	(%0000000x)
	$02 - $05	Starting Logical Blk Addr	(MSB»»LSB)
	$06 - $09	Number of Blocks	(MSB»»LSB)
	$0A	Reserved.	
	$0B	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition
Reservation Conflict

$80AE - Write and Verify; (Optical Memory Devices) [O]

The Write and Verify command requests that the target device write the data send and then after the write, verify that what is on the media is what the host sent. Refer to the Device and ANSI® documents for information concerning the flags and their usage.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($AE)
	$01	DPO	(%000x0000)
		EBP	(%00000x00)
		BytChk	(%000000x0)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06 - $0A	Reserved	
	$0B	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

Data to be written and verified

Errors:

Good
Check Condition
Reservation Conflict

$80AF - Verify; **(Optical Memory Devices)**
[O]

The Verify command is almost identical to the Write and Verify command except that no data is send or written. the target verifies the data that is on the media. Refer to the Device and ANSI® documents for information concerning the flags and their usage.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($AF)
	$01	DPO	(%000x0000)
		Blank Verify (BlkVfy)	(%00000x00)
		BytChk	(%000000x0)
		RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06 - $0A	Reserved	
	$0B	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

None. The request length specifies the number of blocks to be verified.

Errors:

Good
Check Condition

$80B3 - Set Limits; (Direct Access Devices)
[U] (WORM Devices)
(Read-Only Direct Access)

The Set Limits command defines the range within which subsequent linked commands may operate. A second Set Limits command may not be linked to a chain of commands in which a Set Limits command has already been issued.

A read inhibit (RdInh) bit of one indicates that read operations within the range are inhibited. A write inhibit (WrInh) bit of one indicates that write operations within the range are inhibited.

The logical block address specifies the starting address for the range. The number of blocks specifies the number of blocks within the range. A number of zero indicates that the range shall extend to the last logical block on the logical unit.

Any attempt to access outside of the restricted range or any attempt to perform an inhibited operation within the restricted range shall not be performed.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($B3)
	$01	Read Inhibit (RdInh)	(%000000x0)
		Write Inhibit (WrInh)	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06 - $09	Number of Blocks	(MSB»»LSB)
	$0A	Reserved.	
	$0B	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$80BD - Update Block; (**Optical Memory Devices**)
[O] (**WORM Devices**)

The Update Block command logically replaces data on the media with new data. Please refer to the vendors device spec and the ANSI® documents for details about the implementation of this command.

The Command Data structure is defined as:

Byte	$00	SCSI Command Number	($BD)
	$01	RelAdr	(%0000000x)
	$02 - $05	Logical Block Address	(MSB»»LSB)
	$06 - $0A	Reserved	
	$0B	Vendor Unique	(%xx000000)
		Reserved	(%00xxxxxx)

Request Length:

$00000000 - $00xxxxxx (Bytes)

Transfer Length:

$00000000 - $00xxxxxx (Bytes)

Buffer Data Structure:

Data to be used in update.

Errors:

Good
Check Condition

$80C0 - Eject Disk; (Ruby Drive)

The Eject Disk command instructs the target device to eject the media if removal is allowed.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($C0)
	$01	Immed	(%0000000x)
	$02 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$80C8 - Audio Track Search; (Ruby Drive)

The Audio Track Search command provides a means for positioning the optical pickup at address specified by the Search Address parameter. This command returns the status byte when the requested search address is found.

A Play bit of zero indicates the target will enter the hold track state (i.e. pause) when Search Address is found. A Play bit of one indicates the target will output the audio channels in the specified Play Mode when the address is found.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($C8)
	$01	SCSI Command Flags	($00)
	$02	Play Flag	(%000x0000)
	$03	Play Mode	($00 - $0F)
	$04 - $07	Search Address	(MSB»»LSB)
	$08	Address Type	(%xx000000)
	$09 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$80C9 - Audio Play; (Ruby Drive)

The Audio Play command request that the target position the optical pickup at the Playback Address specified and output the audio channels in the specified play mode when and if the Playback Address is located. The status is returned when the Playback Address has been found or if it is not found or if the target is not yet ready to accept the Audio Play command.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($C9)
	$01	SCSI Command Flags	($00)
	$02	StopAddr	(%000x0000)
	$03	Play Mode	($00 - $0F)
	$04 - $07	Playback Address	(MSB»»LSB)
	$08	Address Type	(%xx000000)
	$09 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$80CA - Audio Pause; (Ruby Drive)

The Audio Pause command temporarily stops the audio play operation and enters the hold track state (i.e. keeps the media at the same Q Subcode address).

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($CA)
	$01	SCSI Command Flags	($00)
	$02	Pause Bit	(%000x0000)
	$03 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$80CB - Audio Stop; (Ruby Drive)

The Audio Stop command causes the target to stop the audio play operation at the Stop Address. The media will spin down and the optical pickup will be held near the area of the stop address.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($CB)
	$01	SCSI Command Flags	($00)
	$02 - $05	Stop Address	(MSB»»LSB)
	$06	Address Type	(%xx000000)
	$07 - $0B	Reserved.	

Request Length:

Unused

Transfer Length:

Unused

Buffer Data Structure:

None

Errors:

Good
Check Condition

$80CD - Audio Scan; (Ruby Drive)

The Audio Scan command requests that the target device transfer the current audio play status and the starting Q Subcode address of the next track to the host computer.

The **Command Data** structure is defined as:

Byte	$00	SCSI Command Number	($CD)
	$01	SCSI Command Flags	($00)
	$02 - $05	Scan Address	(MSB»»LSB)
	$06 - $0B	Reserved.	

Request Length:

$00000000 - $000000ff (Bytes)

Transfer Length:

$00000000 - $000000ff (Bytes)

Buffer Data Structure:

See Device and ANSI® Documents for parameter block descriptions.

Errors:

Good
Check Condition

Driver Flush

Call Parameters :	Device Number	≠	$0000
	Call Number	=	$0007
	DIB Pointer		

Device Number: *This word parameter specifies the target device. This parameter must be nonzero.*

Call Number: *This word parameter specifies the type of call.*

DIB Pointer: *This longword points to the DIB for the target device.*

This call is issued only in preparation for a close or shutdown call. A character device which maintains it's own buffer will output any portion of the contents of that buffer which has not already been output to the device. A driver that does not maintain it's own data buffers will take no action. This call is not supported by block device drivers and should return a 'BAD COMMAND' error.

Driver Shutdown

Call Parameters :	Device Number	≠	\$0000
	Call Number	=	\$0008
	DIB Pointer		

Device Number: *This word parameter specifies the target device. This parameter must be nonzero.*

Call Number: *This word parameter specifies the type of call.*

DIB Pointer: *This longword points to the DIB for the target device.*

This call is issued by GSOS in preparation for purging the driver and will execute any operation necessary in preparation for purging the driver. This includes releasing memory back to the memory manager that was previously acquired for buffer usage. If the device is open, a close call should be issued to the device. **This call may not be issued by either an application or an FST!!!**

Communicating with the Device

Please refer to the SCSI Manager External ERS for the details of communicating with the devices.

Device Driver Error Codes

All error codes listed below must be supported by device drivers wherever applicable. All block device drivers must support disk switched errors without exception. Please take note that the error codes are returned from a device driver must have the high byte cleared. The device dispatcher maintains certain error codes under certain conditions. Device dispatcher error codes are passed in the upper byte of the accumulator.

Error Code	Description	Mnemonic
$0000	No error occurred	NO_ERROR
$0010	Device not found	DEV_NOT_FOUND
$0011	Invalid Device Number	INVALID_DEV_NUM
$0020	Invalid request	DRVR_BAD_REQ
$0021	Invalid control or status code	DRVR_BAD_CODE
$0022	Invalid parameter	DRVR_BAD_PARM
$0023	Device not open (character driver only)	DRVR_NOT_OPEN
$0024	Device already open (character driver only)	DRVR_PRIOR_OPEN
$0026	Resource not available	DRVR_NO_RESRC
$0027	I/O error	DRVR_IO_ERROR
$0028	Device not connected	DRVR_NO_DEV
$0029	Device is busy	DRVR_BUSY
$002B	Write Protected (block driver only)	DRVR_WR_PROT
$002C	Invalid Byte Count	DRVR_BAD_COUNT
$002D	Invalid Block Number (block driver only)	DRVR_BAD_BLOCK
$002E	Disk Switched (block driver only)	DRVR_DISK_SW
$002F	Device Off Line or No Media Present	DRVR_OFF_LINE
$004E	Invalid access or access not allowed	INVALID_ACCESS
$0058	Not a block device	NOT_BLOCK_DEV
$0060	Data is unavailable	DATA_UNAVAIL

The Apple IIGS Installer V1.1

by

"Jay" Schaffer

NOTE: PAGE 2 REMOVED INTENTIONALLY.
NO TECHNICAL INFORMATION IS MISSING.

TABLE OF CONTENTS

1 HISTORY

Until the Apple IIGS Installer Program V1.0 was written, there had never been a *standard* installer program for the Apple II family of computers. Version 1.1 of that program endeavors to make additional enhancements to the original program and correct any bugs that were discovered after it was released.

2 OBJECTIVES

There are many instances when simply copying a file from one volume to another is not sufficient to install system or application software. Typically a number of tools should be updated simultaneously along with new patches, etc. The object of the original project was to develop a standard Apple IIGS installer program which operated as a script interpreter for adding, replacing or deleting files on any readable/writeable ProDOS® or AppleShare® volume with sufficient parameter passing to make it as flexible as possible. And, as mentioned above, version 1.1 is intended to offer additional enhancements and to fix any bugs discovered after the release of the original program.

The major enhancements fall into the following areas:

- Boot Code Installation - The Installer can now read in a 1024 byte file and use it to replace the boot code on any ProDOS® volume that allows block writes.

- Multiple Script Selection - The user may now select more than one script for execution before *pressing* INSTALL or REMOVE.

- Improved Error Reporting - When errors associated with script syntax or failure to locate files specified within scripts occur, the error dialog will now report the source and destination names found in the file specification so that script writers may locate the offending FileSpec field quicker.

- Network Installation - The Installer program can make installations over networks.

Many other totally transparent changes have also been made in the code, but because of their transparency, they are not mentioned in this document.

3 OVERVIEW

GS/OS™ V3.0 is required for the Apple IIGS Installer V1.1 which is a general purpose GS/OS™ utility program that is designed to install, update and remove files from volumes that can be read and written by GS/OS™. The Installer is controlled by "scripts" which contain lists of files along with other information. The scripts are located in a SCRIPTS subdirectory in the same directory as the Installer program. Multiple scripts may be present in the SCRIPTS subdirectory and the user may select more than one script at a time for execution . Files available for copying may be located anywhere on any volume. In many cases, it may be advantageous to place the files to be copied in a subdirectory located at the same level as the Installer program itself. Different scripts can have different subdirectories of files so that they can handle different files with the same name.

When the Installer is executed, the display will include a list of all the scripts in the SCRIPTS subdirectory for the user to select from and provisions for asking for help. System file update scripts will normally ask only for the name of the volume to be updated. A "*Disk*" button will let you step through all the online volumes until the desired one is found. Other scripts will commonly require the name of the directory where the update is to be installed.

When multiple scripts are selected, they will be executed in the same order that they appear in the update selection window. It is important to note several important facts about script execution:

- Each script is processed from beginning to end the same as if it were the *only* script selected.
- If boot code replacement is requested, it *must* be the first FileSpec and it can only take place on a volume that accepts _Block_Write calls. If the _Block_Write calls to write the boot code file to the boot blocks fails, *no* error trapping will occur, however, the remainder of the updating will continue as if *everything* had gone just fine.
- If the execution of a script generates an error, or if the user terminated further processing of a script, the queue will be cleared of any additional scripts waiting to be excuted and control will return to the user.
- It is possible for several scripts to execute successfully before one is encountered which cannot be executed because of insufficient free space remaining on the destination volume.

Script files will be read into memory in their entirety, then parsed, stripped of all comments, and compacted into the least possible space and verified. Following compaction, scripts will be executed in two passes. The first pass will determine whether there is sufficient space on the volume for the desired update to be accomplished. If not, execution will terminate and the user will be notified. Additional space required is reported by first determining the number of blocks needed, dividing by two and reporting as additional (1024 byte) kilobytes needed.

Please note that it is *totally impossible* to accurately determine block requirements of directories. The Installer's space calculation algorythms are *pretty* good but cannot be guaranteed to be exactly correct!

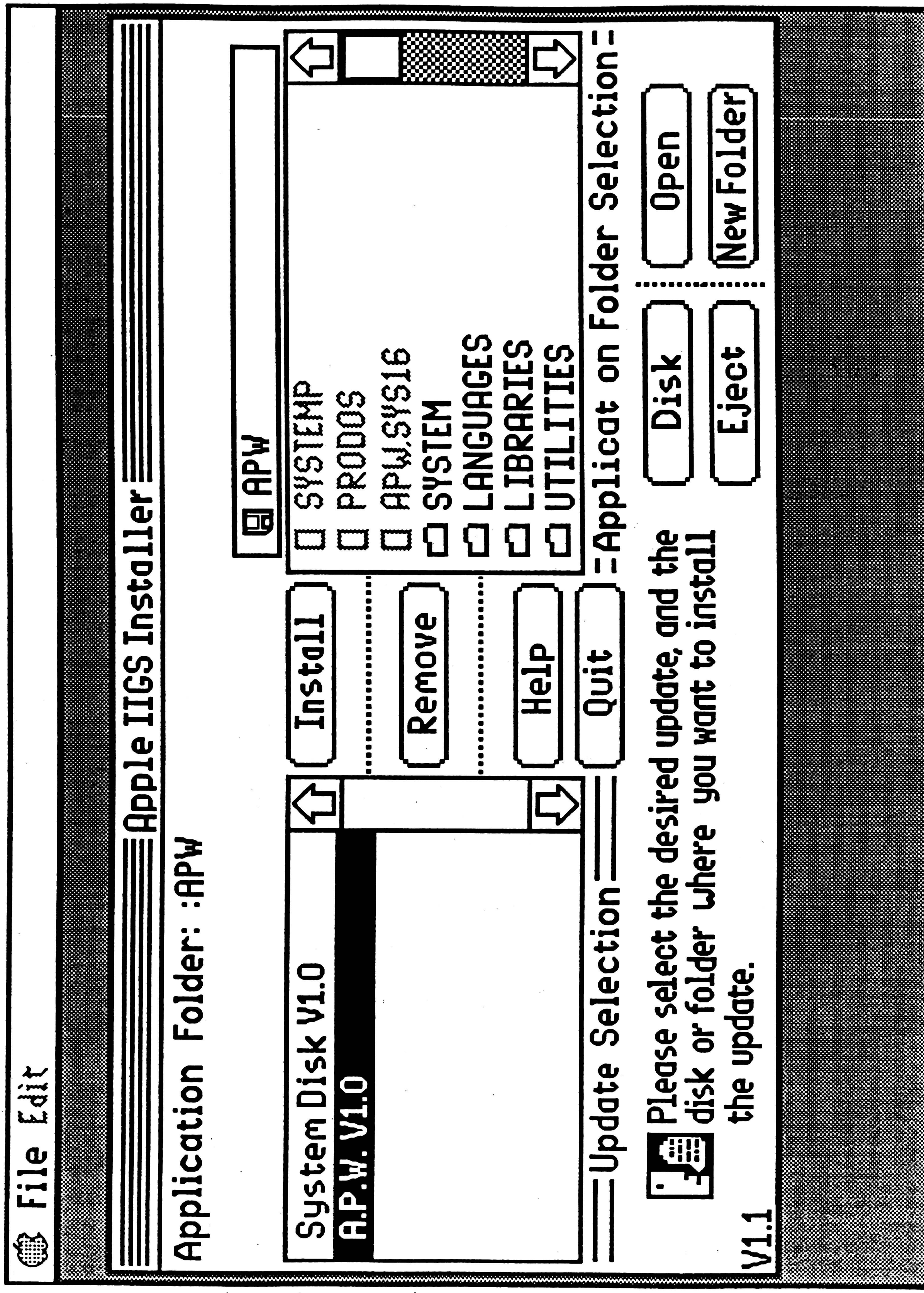
If the Installer's calculation determine there is room, the installer will "*read*" the selected script a second time, deleting and copying files in accordance with special instructions passed from the script in flags. Locked destination files will be *"blindly"* unlocked. Also, the Installer will *"blindly"* replace all files requested by the script without regard to the versions or creation dates of existing files on the user's volume. In many cases, files will be replaced by identical files. However, due to the lack of version information and the manner in which date stamping of files is handled under ProDOS® and GS/OS™, there is no way to know with certainty about a files real date or version.

Installation or Removal of system files to/from the boot disk is a special case. Replacing or removing tools from the boot disk may create problems for old memory resident tools which may not operate with a newly replaced tool loaded from disk. In addition, replacing the SYS.RESOURCES file could create other problems of it's own. Therefore, whenever a system level update occurs (one which does not require application folder selection) involving the boot disk, the Installer will disable all desk accessories and close the SYS.RESOURCES file. Any attempt to "Quit" from the Installer program will bring forth a dialog explaining that system files have been replaced on the boot disk and that it is necessary to re-start (reboot) the system. The default response will be to re-start.

After processing the selected script(s) is completed, a dialog will notify the user that the "Installation" or "Removal" has been successfully completed.

4 DETAILED DESCRIPTION

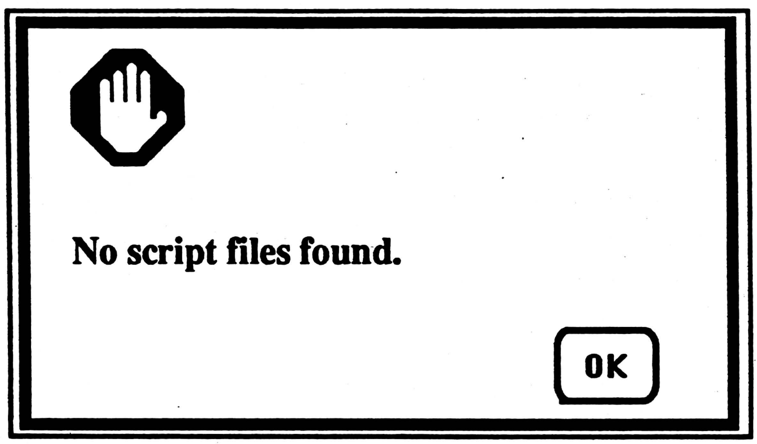
4.1 The User Interface



The Apple IIGS Installer program *requires* GS/OS V3.0 or later. If it is launched under ProDOS16 or earlier versions of GS/OS, the user will be notified of the requirement and the program will terminate.

During the startup process, all needed tools are loaded and started up. If an error occurs before we can even present a desktop, The System Death Manager will be called to display the error.

As the startup and initialization process continues, a search is made for script files. If there is no SCRIPTS subdirectory in the same directory as the Installer program, if no script files can be found in the SCRIPTS subdirectory, or if attempted access to the SCRIPTS subdirectory or all script files results in an error, an alert will notify the user that the Installer program cannot continue.



If scripts can be found and accessed, the Installer program will continue and the *Main Selection Screen* will be displayed.

Figure 1 shows the Apple IIGS screen once the Installer program is successfully "up and running". The contents of the screen may be divided into three groups: (1) "update" selection and execution controls, (2) "disk" selection controls, and (3) directory (folder) selection controls.

4.1.1 Update Selection and Execution Controls

To make the installer program more friendly to computer novices, the scripts are refered to as "updates" or "update scripts" in the user displays. However, in this document, we describe them as scripts for the more knowlegeable reader.

4.1.1.1 Update Selection Window

Each script found in the SCRIPTS folder in the same directory as the Installer program is opened. If it begins with the characters "SCRIPT" followed by a compatible ScriptVersion number, it's *Script Name* field is read and displayed in a scrolling window. The first entry is initially high-lighted by default. If there are too many script names to be displayed, the vertical scroll bar becomes active permitting scrolling through the list.

At startup, each script is opened and it's ScriptName field is read into a buffer for display, it's filename is stored for possible later retrieval, and information is stored indicating whether the script permits a *Remove* command and whether it needs to be told the location of the files to be updated. This information is used to alter the active/inactive status of controls associated with application folder selection based on the currently selected (hilited) script(s).

The Update Selection Window is controlled by the *List Manager*. The List Manager allows the user to choose between three possible selection modes: single, arbitrary, and range. The Apple and Shift keys are used to choose the selection mode. The state of the Apple and Shift keys is checked only when the user first presses the mouse button. After that, the user can release the key, and the selection mode remains in effect until the user releases the mouse button. The three modes are as follows:

- Single mode: The user selects single mode by simply pressing the mouse button and not pressing either the Apple or Shift key. Any selection the user makes deselects all other selected members. Thus, when the user drags the mouse, the selection moves from one member to another.

- Arbitrary mode: If the user holds down the Apple key and then presses the mouse button, already selected members are not deselected. This allows unselected members to be between selected members in the list. Dragging is allowed in this mode, so any enabled member the mouse is dragged over will be selected. The arbitrary mode overrides the range mode if the user presses both the Apple and Shift keys.

- Range mode: If the user holds down the Shift key and then presses the mouse button, a range of members is selected, and all of the members outside the range are deselected. A range is defined as follows: The first selected member in the list is the beginning of the range the end of the range is the current selection if it appears after the beginning of the range. If the current selection is the first selection in the list, and therefore the beginning of the range, then the end of the range is the last selected member in the list.

Any number of scripts may be selected by using one or more of the selection modes described above. The high-lighted script name(s) is considered the currently selected script(s). When more than one script is selected, the *Help* button will be disabled and the *Remove* button will only be active if *all* of the selected scripts permit the Remove function.

4.1.1.2 Install Button

Clicking on the *Install* button will cause the highlighted script(s), in the "scrolling window", to be executed, installing or replacing files on the selected volume according to the script's instructions. Each selected script will be executed in the same order that they appear in the update selection window.

If a script calls for system level operations (application folder selection is <u>not</u> required) on the boot disk, <u>all</u> desk accessories will be disabled, the SYS.RESOURCES file will be closed, and the user will no longer be able to *Quit* back to the application that launched the Installer (usually the Finder).

4.1.1.3 Remove Button

If the *Remove* button is active, clicking on the *Remove* button will cause the highlighted script, in the "scrolling window", to be executed, removing files from the selected volume according to the script's instructions.

The *Remove* button may be inactive because the currently selected script does not permit a Remove function, or one or more of the set of currently selected scripts does not permit a Remove function.

4.1.1.4 Help Button

If the *Help* button is active, clicking on the *Help* button will cause the contents of the *ScriptHelp* field of the hilited script to be displayed in a dialog box. Information about the Installer program will also appear. The dialog box will contain an *OK* button which may be clicked, or Return or Enter may be pressed to return to the *Main Selection Screen*.

They *Help* button will be inactive when ever more that one script has been selected.

4.1.1.5 Quit Button

If no script(s) is currently being executed and if no system level operations have been performed on the boot disk, clicking the *Quit* button, pulling down the *File* menu and selecting *Quit*, or pressing ⌘-Q or ⌘-q will execute a GS/OS™ QUIT call, terminating the Installer program.

If no script(s) is currently being executed and if system level operations have been performed on the boot disk, the user will be presented with a dialog that explains that the system must be re-started. The two possible responses wil be to "Restart System" (the default response) and "Cancel" which will return the user to the Installer Program.

4.1.2 Disk Selection Controls

The work "Disk" is used rather the the work "Volume" on the control button in hopes of making the installer program more friendly to computer novices. It should be noted that partitioned volumes will appear as multiple disks to the user. This document, however, still describes everything in terms of "Volumes" for clarity to the knowledgeable reader.

4.1.2.1 Disk/Folder to be Updated

A "static text" item appears left-justified and directly below the Installer Program's window title bar which identifies the pathname of the directory (folder) which will receive the desired update.

If the currently selected script is intended to update the system files on a system startup disk, the static text item will read "Disk to Update:" followed by the volume's name.

If the currently selected script is intended to update an application program, requiring that the Installer program be told where the application is located, the static text items will read "Application Folder:" followed by the pathname of the currently displayed directory. If the full pathname of the currently displayed directory is too long to fit on a single line, then the display shall be shortened by displaying the volume name, three periods, and as many levels of the end of the full pathname as well fit on the line.

If multiple scripts are selected, all of which are intended to update the system files on a system startup disk, the static text item will read "Disk to Update:" followed by the volume's name.

If multiple scripts are selected and one or more are intended to update an application program, requiring that the Installer program be told where the application is located, the static text items will read "Application Folder:" followed by the pathname of the currently displayed directory. If the full pathname of the currently displayed directory is too long to fit on a single line, then the display shall be shortened by displaying the volume name, three periods, and as many levels of the end of the full pathname as well fit on the line.

4.1.2.2 Disk Button

The *Disk* button is used for selecting the volume to be updated. Pressing the *TAB* key will provide identical functionality as clicking the *Disk* button. Each time the *Disk* button is clicked or the *TAB* key is pressed, the name of the next volume in the GS/OS™ device list will be displayed as the volume or folder to be updated. After the last volume name is displayed, the next click will result in "wrap-around" and the volume name of the first device in the list will be displayed. Only devices that support read and write will be acknowledged, including networks

If (one or more of) the currently selected script(s) is intended to update an application program, requiring that the Installer program be told where the application is located, the contents of the volume directory will appear in the *Directory Selection* window and the volume name will appear as the title of an empty menu in the pop-up menu bar above the *Directory Selection* window.

4.1.2.3 Eject Button

The *Eject* button permits ejecting disks when appropriate. If the drive contains removable media and is NOT a Disk][, Unidisk, Duodisk, or Disk //c, the eject request will be sent to the operating system. Although this is not 100% certain to work under every circumstance, it is the best that can be obtained and should be very close to fool-proof.

It is important to note that AppleShare volumes will appear as removable media. *Eject*ing an AppleShare volume will log the volume off and remove it from the device list.

4.1.3 Directory (Folder) Selection Controls

The directory (folder) selection controls become inactive if (one or more of) the currently selected script(s) is intended for system disk updating. If (one or more of) the currently selected script(s) indicates that the user must indicate where an update is to take place, these control come to life.

4.1.3.1 Directory Selection Window

This scrolling window is used to notify the Installer program where an update is to take place. If the currently selected script(s) is only intended to update the system files on a system startup disk, this window will be blank. This window supports only single seletion mode (see 4.1.1.1).

The scrolling window appears as a normal *Standard File* type window. The current directory is displayed at the top of the window as the title of a pull-down menu. The window lists all of the files contained in the current directory. Only directory files are hilited and active. Other files in the current directory are inactive and "dimmed-out". If the first file is a directory file, it will be hilited by default. The tree-like directory structure may be traversed by *Open*ing directory files until the directory that is to receive the update appears as the title of the *Current Directory Menu* at the top of the window.

4.1.3.2 Current Directory Menu

Directly above the *Directory Selection* window is a *pop-up* menu whose title is the currently displayed directory (folder) name. This is the mechanism used for closing a directory and backing up in the volume's tree-like directory structure.

When the menu is pulled down, the name of each directory (folder) in the path to reach the currently displayed directory will appear as an entry with the volume name listed last at the bottom of the menu. If the currently displayed directory (folder) is the volume directory, there will be no items in the menu—it will not pull down and the title will be the volume name.

Selecting a directory (folder) in the pull down menu will result in that directory (folder) becoming the currently displayed directory and it's name will become the new menu title.

If the currently selected script(s) is intended to update the system files on a system startup disk, this "menu bar" will display the volume name.

4.1.3.3 Open Button

This button is only active if (one or more of) the currently selected script(s) is intended to update an application program, requiring that the Installer program be told where the application is located. It will open the hilited directory (folder), making it the current directory and displaying it's contents in the *Directory Selection* window. The same functionality may be accomplished by "double clicking" on the desired directory name in the *Directory Selection* window.

4.1.3.4 New Folder Button

This button is only active if (one or more of) the currently selected script(s) is intended to update an application program, requiring that the Installer program be told where the application(s) is located. It will present a dialog box with a place to enter the name of the new folder to be created. Any combination of up to 32 "printable" characters will be accepted. The name will be passed on to GS/OS to see if the name is sytactically correct, since different FSTs have differing rules of syntax. Once the name has been accepted, the new folder will appear in the currently displayed directory (folder). If the name is not accepted, the user will be notified of the reason and an opportunity to enter another name will be given.

4.1.4 Script Execution

It is important to note that the Apple IIGS Installer program makes no attempt to identify versions of files to be updated through creation dates or any other technique. It is felt that there is no truly reliable way to handle this problem. Therefore, script instructions are "blindly" followed even to the point of replacing existing files with identically the same file.

Once the user has selected the script(s) to execute and the destination volume and/or directory (folder), execution of the first script may commence. All that is required is to click the *Install* or *Remove* button while they are active. Scripts are executed in the same order as they appeared in the update selection window.

When the script is read into memory, the ScriptFlag is checked to see if a Caution Alert should be displayed. Since some scripts represent installations that would be <u>totally</u> in appropriate on certains volumes, this facility permits the script writer to *force* the user to read the script's help message before beginning any file manipulations associated with the script. At this point the user has a choice to continue with the script or to skip it and continue with the next script, if any. If the user wishes to execute the script, it is parsed, verified, and formatted for our use, with the header and all comments removed. If more than one script has been selected that requires application folder selection, all those scripts will use the one and only folder that has been selected as the *Application Folder*.

The first pass reads the script instructions and calculates the size of all the files to be deleted and the size of all the files to be copied and then determines, as best it can, if there is sufficient room on the destination volume to accept the <u>complete</u> update. No updating will commence unless the Installer's calculation had determined that there is sufficient room since this could ultimately create problems with an unknown mix of file versions. If there is insufficient space, the user will be notified and the program will terminate execution of the current script and clear the queue of any addtion scripts waiting to be executed. The number of additional blocks required is divided by two to obtain the number of kilobytes that are needed. The user must then make room on the volume through file deletion if it is desired to attempt the same installation again on the same volume.

Once it has been verified that there is sufficient room for the <u>complete</u> update to take place, a second reading of the script takes place. If boot code is to occur, it's FileSpec <u>must</u> to first! If the destination volume will not accept _Write_Block calls (such as a network or non-ProDOS volume), the boot code will *not* be installer, however, *no* error will be generated.

Following boot code replacement, if any, files to be replaced/removed on the destination volume are "blindly" unlocked and all file deletions and file copying associated with the update process takes place. Any file that is too large to fit into available memory will be transferred in several passes. If required subdirectories are missing, they will be created so that files may be copied into them.

The user may terminate execution of a script and any scripts which follow by pressing ⌘-.. A check for key-events at the end of the first pass and between each file transfer. If the user requests termination of the script, they will be warned of the possiblility of an unknown mix of files on the destination and given the opportunity to continue or halt execution (see 4.1.5.1).

Typically, a script will be written in such a manner that if the user added all the files to the wrong directory, he may re-execute the script, clicking the *Remove* button to remove everything, except the folders which may (or may not) have been created during the installation. However, there is no way to restore files that may have been deleted during the installation process.

After all processing of the instructions in a script is sucessfully completed, a check will be made to see if additional scripts were also selected. Each script selected will be executed in succession until there are no more to process. Finally, the Installer program will return to the main selection screen, giving the user an opportunity to once again make selections and perform other operations.

It is important to once again note several important facts about script execution:

- Each script is processed from beginning to end the same as if it were the *only* script selected.

- If the execution of a script generates an error, or if the user terminated further processing of a script, the queue will be cleared of any additional scripts waiting to be excuted and control will return to the user.

- It is possible for several scripts to execute successfully before one is encountered which cannot be executed because of insufficient free space remaining on the destination volume.

- If more than one script has been selected that requires application folder selection, the one and only *Application Folder* that the user has selected will be used by all of those scripts.

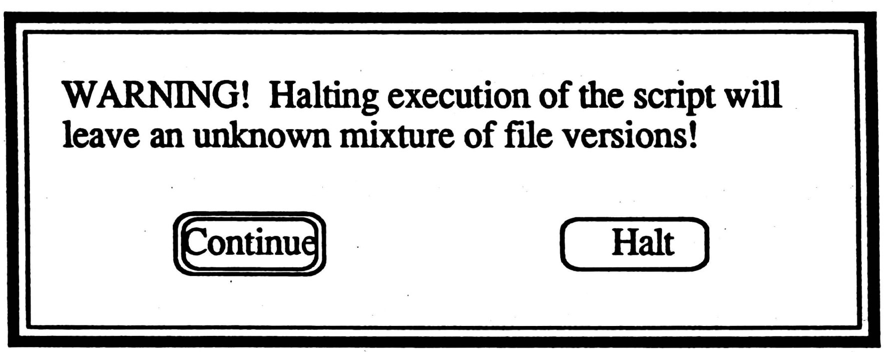
It should also be remembered that Installation or Removal of system files to/from the boot disk is a special case. Replacing or removing tools from the boot disk may create problems for old memory resident tools which may not operate with a newly replaced tool loaded from disk. In addition, replacing the SYS.RESOURCES file could create other problems of it's own. Therefore, whenever a system level update occurs (one which does not require application folder selection) involving the boot disk, the Installer will disable all desk accessories and close the SYS.RESOURCES file. Any attempt to "Quit" from the Installer program will bring forth a dialog explaining that system files have been replaced on the boot disk and that it is necessary to re-start (reboot) the system. The default response will be to re-start.

After processing the selected script(s) is completed, a dialog will notify the user that the "Installation" or "Removal" has been successfully completed.

4.1.5 Error Handling

4.1.5.1 User Cancel Request

If, after script execution has commenced, the user presses ☐-., in effect they have created an error condition since they now have an unknown mix of file versions. A dialog box will appear giving an opportunity to continue or halt script execution. A user cancel request will not be acknowledged until the current file copy or deletion is completed.



Terminating execution of a script will also clear the queue of any other scripts waiting to be executed and return control to the user.

4.1.5.2 Non-Recoverable Errors

Some errors will simply be fatal. If a file is corrupted, the media is bad, or directories are corrupted such that a deletion and/or insertion of a file cannot take place, execution of the script will terminate, and the user will be notified of the error that occurred in a *Stop Alert* box.

Clicking on the *OK* button will cause the queue of other scripts waiting to be executed to be cleared and the Installer program to return to the *Main Selection Screen.*

4.1.5.3 Script Errors and File Not Found Errors

When ever the Installer program detects a script error or a File Not Found error, it will report the name of the source file and destination file it was most recently processing along with the normal error message. This should aid script writers in going directly to the offending FileSpec field to make corrections. If the error was associated with the header, no file name will be reported. As with all errors, the queue of other scripts waiting to be executed will be cleared and the Installer will return to the *Main Selection Screen.*

4.1.5.4 Volume Not Found Errors

Volume Not Found errors will bring up a dialog asking the user to insert the missing volume. If "OK" is selected, the file access call will be attempted again. If "Cancel" is selected, an error will be flagged and the Installer will return to the main selection loop. As with all errors, the queue of other scripts waiting to be executed will be cleared and the Installer will return to the *Main Selection Screen.*

4.2 Script Files

Script files are a simple ASCII (type TXT) file and must be located in a subdirectory (folder) named SCRIPTS located in the same directory as the Installer program. Scripts may not exceed 65535 bytes in length. An attempt to execute a script that is too large will result in a "Stop" alert.

A script is a list of instructions for the installer. A script can specify that files be copied from a "source" volume to (or removed from) a "destination" volume (or folder, when applicable) that the user chooses.

The Script consists of a header field followed by any number of file specification and/or comments fields. The field separator is a tilde (~) character. Most entries within a field are separated by <return> characters. Two consecutive tilde characters designates the end of the script. Any additional characters past this point are ignored.

4.2.1 Header Field

The Header Field consists of ASCII characters, hi-bit clear, as defined in the following sections. No comments are permitted in any of the following entries.

4.2.1.1 Script File Identifier:

To be recognized as a Script File, the Header Field must commence with:

SCRIPT<return><return>

The eight hexadecimal bytes are:

53 43 52 49 50 54 0D 0D

4.2.1.2 ScriptVersion

The next 7 bytes define the minimum version of the Installer Program that can read and execute the instruction in the Script File. Scripts which move files containing resource forks ***MUST*** be designated as V1.10, as shown below, since the original Installer will only move the data fork and no error will be returned! Others may continue to be designated as V1.00 for compatibility with the original release of the Installer program.

V1.10<return><return>

The seven hexadecimal bytes are:

56 31 2E 31 30 0D 0D

4.2.1.3 ScriptFlag

The next 4 bytes define the directory requirements of the Script File. The first character must be either a "R" (indicating that the installation must occur at the root directory, such as in a system disk update), or "X" (indicating that the user must specify the directory where installation should take place).

The second character must be either a "R" (indicating that the *Remove* command is valid for this script) or a "N" (indicating that the *Remove* command is not valid and the button should be dimmed and inactive). If the second character is lower case, before any files manipulations commence the Installer program will display a *Caution Alert* filled with the contents of the ScriptHelp field and button controls to permit the user to choose whether to execute the script or skip it and go on to the next script, if any.

As an example of a ScriptFlag, these four bytes may be:

RR<return><return>

The four hexadecimal bytes are:

52 52 0D 0D

4.2.1.4 ScriptName

The next series of bytes defines the name of the script as it will be displayed in the Update Selection window by the Installer Program. For appearance's sake, it is recommended that care be taken to use a name that will fit within the display window. This entry consists of ASCII characters ending with a <return> character. It may not include a tilde or <return> character.

4.2.1.5 ScriptHelp

The next series of bytes defines the text to appear when the Help function is selected. This entry consists of ASCII characters ending with two backslashes and a <return> and may not include two consecutive backslashes or a tilde character. For appearance's sake, it is recommended that care be taken to use text that will fit within the display window. This entry may not include two consecutive backslash characters or the tilde character. It may include <return> characters.

4.2.1.6 SourcePrefix

The last entry, if any, is a prefix to be used with source files defined by partial pathnames. The SourcePrefix name must be terminated by a FieldSeparator (tilde). If there is to be no SourcePrefix, this entry must be null, and the <return> following the two backslashes which terminates the ScriptHelp entry must be immediately followed by a FieldSeparator (tilde). Either slashes or colons may be used as the pathname separator character.

If no SourcePrefix is specified, the name of the volume from which the installer program is running shall become the SourcePrefix.

4.2.2 File Specification Field

The Script may include as many file specification fields as needed, each preceded by a tilde character to separate it from the following field. The tilde must be immediately followed by the first FileSpec Flag and may not be immediately followed by a <return> character.

4.2.2.1 FileSpec Workspace

Each File Specification Field shall begin with a 16-character workspace. Any character may be used except a tilde and it may not begin with an asterisk. It is our suggestion that fifteen readable characters followed by a <return> might be easiest to see and count, such as:

:::Workspace:::*<return>*

4.2.2.2 FileSpec Flags

There may be as many FileSpec Flags as needed. Each must appear at the beginning of a line, except the first which must immediately follow the 16-character workspace (the last character of which may be a *<return>*). FileSpec Flag characters need not be immediately followed by a <return> character, however, and in this case, all characters between the FileSpec Flag character and the <return> character will be ignored. Thus it is possible to place comments on the same line with a FileSpec Flag. The final flag, whether it is one of the required flags or one of the optional flags must be followed by two <return> characters.

4.2.2.2.1 Required Flags

One, and only one,of the following flags MUST be present among the FileSpec Flags.

1 If the user clicks "Install" then delete the file from the destination volume and copy the file from the source volume.
If the user clicks "Remove" then delete the file from the destination volume if it exists.

2 If the user clicks "Install" then delete the file from the destination volume and copy the file from the source volume.
If the user clicks "Remove" then ignore it.

3 If the user clicks "Install" then delete the file from the destination volume if it exists.
If the user clicks "Remove" then delete the file from the destination volume if it exists.

4 If the user clicks "Install" then delete the file from the destination volume if it exists.
If the user clicks "Remove" then ignore it.

4.2.2.2.2 Optional Flags

One or more of the following flags may be present among the FileSpec Flags:

B The file designated by the source pathname is boot code to replace blocks zero and one on the destination volume. (This flag must be used with a "2" flag from above and any destination pathname will be ignored.) **Important:** Boot code replacement **must** be the first FileSpec in the script!

C The creation date and time of the file designated by the source pathname must match the CreationDate entry in the FileSpec field.

D The designated destination file should be deleted if, and only if, it has a creation date/time that is older than the supplied creation date/time. (This flag must be used with a "4" flag from above.)

F The file type and auxiliary type of the file designated by the source pathname must match the FileType entry in the FileSpec Field.

U Update (replace) existing destination file only if it exists. (This flag must be used with a "1" or "2" flag from above.)

4.2.2.3 FileType and Auxiliary Type

This entry consists of four ASCII characters...a four-character file type and an eight-character auxiliary type. The twelve character file/aux type entry need not be immediately followed by a <return> character, however, in this case, all characters between the file type characters and the <return> character will be ignored. Thus it is possible to place comments on the same line with the file type characters.

An entry for a file type of "TXT" ($04) and an aux type of $1000 would appear as:

000400001000[optional comment]<return>

The hexadecimal translation would be:

30 30 30 34 30 30 30 30 31 30 30 30[optional comment] 0D

4.2.2.4 Creation Date and Time

This entry consists of fifteen ASCII characters...nine for the date, a space and five for the time. The three-character month abbreviation may appear in any combination of UPPER/lower case. Leading zeros in the date should be replaced with spaces. The creation date/time entry need not be immediately followed by a <return> character, however, in this case, all characters between the creation date/time characters and the <return> character will be ignored. Thus it is possible to place comments on the same line with the creation date/time characters.

A typical entry might appear as follows:

10 Jan 88 23:32[optional comment]<return>

The hexadecimal translation would be:

31 30 20 4A 61 6C 20 38 38 20 32 33 3A 33 32 [optional comment]<return>

4.2.2.5 Source Pathname

The source pathname should describe name and location of the source file by any means that represents acceptable syntax under GS/OS™, the operating system for the Apple IIGS. If the pathname is a partial pathname, the source prefix, above, will be used. If the FileSpecFlags indicate removal only, the source pathname may be a null string (a <return> character only). Either slashes or colons may be used as the pathname seperator character. No optional comments are permitted!

A typical entry might be:

1:ProDOS<return>

4.2.2.6 Destination Pathname

The destination pathname should describe name and location of the destination file by any means that represents legal syntax under GS/OS™, the operating system for the Apple IIGS. Only partial pathnames are normally used—the prefix has already been set by the Installer Program to the location of the destination directory, either the root directory or a user selected directory. If the FileSpecFlags indicate that this file specification is for boot block replacement *(which is not implemented in the initial release)*, there is no destination pathname and this entry may be a null string (a <return> character only). No optional comments are permitted!

A typical entry might be:

Libraries:AInclude:E16.ADB<return>

4.2.3 Comment Fields

Comment fields may be included in script files. They must begin with an asterisk (*) character immediately following the tilde which separates fields. As with all fields, they must be separated from the following field by a tilde (~) character. The contents of Comment fields is ignored.

5 TESTING RECOMMENDATIONS

The Installer Program is sufficiently complex that *total* testing is impossible. The following simple recommendations were offered by the author for version 1.0 and are intended to provide a launching point for a comprehensive testing plan.

1.) Write a number of scripts that deal with only one file. Each script can test a different combination of FileSpec Flags.

2.) Write a number of scripts that deal with installations up to and exceeding the maximum depth of installation. Try them on newly initialized volumes as well as ones that have previously received the same installation or removal.

3.) Try bounds testing with a few very large scripts dealing with a multitude of files. Use volumes upon which the installation will just fit and will not quite fit. (Remember, the reporting of additional space required is not guaranteed to be exact!)

4.) Test the *Main Selection Screen* by trying all legal and illegal operations with and without the proper volumes on line.

In addition, the following additional test should be included to test the new features included in version 1.1:

5.) Try installing boot code under legal and illegal conditions.

6.) Try various combinations of multi-script selections.

6 SCRIPT EXAMPLE

Scripts may be created with any ASCII text editor. They are of type TXT ($04).

The following simple example file will demonstrate the proper format of a script file. The RETURN characters are shown as <return> for clarity.

```
SCRIPT<return>
<return>
V1.00<return>
<return>
RR<return>
<return>
Example Text Script V1.0<return>
This is an example script only.  It serves no actual purpose, however. it will
operate and install files as directed by the information in this script.\\<return>
~*  (No Source prefix in header)<return>
<return>
--- End of Script File Header ---<return>
<return>
~:::Workspace:::<return>
3            Install if INSTALL clicked but ignore if REMOVE clicked<return>
U            Update only if destination file exists, don't add the file<return>
<return>
<return>
<return>
1:ProDOS<return>
ProDOS<return>
~*<return>
<return>
---  End of 1st file specification ---<return>
<return>
~:::Workspace:::<return>
3            Install if INSTALL clicked but ignore if REMOVE clicked<return>
U            Update only if destination file exists, don't add the file<return>
C            Be sure source file is right one by matching creation date<return>
F            Be sure source file is right one by matching file type<return>
<return>
00FF00000000<return>
03 Sep 87 22:36<return>
1:SYSTEM:P8<return>
SYSTEM:P8<return>
~*<return>
<return>
---  End of 2nd file specification ---<return>
<return>
~~*          Double tilde marks the end of the script file.
```

Appendix A.

Error Messages

The following error messages are generated by the Installer program when the indicated error occurs.

ProDOS® - GS/OS™ Errors:

$27	I/O Error
$28	No Device Connected
$2B	Disk Write Protected
$2E	Disk Switched before FLUSH
$40	Invalid Pathname Syntax
$43	Invalid reference number
$44	Path not found
$45	Volume Directory not found
$46	File not found
$47	Duplicate File Name
$48	Volume Full
$49	Volume Directory Full
$4A	Incompatible File Format
$4B	Unsupported storage type
$4C	EOF has been encountered
$4D	Position out of range
$4E	Access error. Cannot change file
$4F	Buffer too small
$50	File is already open
$51	Directory is Corrupted
$52	Not a ProDOS disk
$57	Duplicate Volume online

Installer Program Errors:

$81	Out of memory.
$82	No script files found.
$83	Pathname is too long.
$84	Script File is too big to handle.
$85	No End-of-Script mark found.
$86	Bad Script File format.
$87	Wrong source file(s).
$88	Cannot INSTALL. Need approximately *nnnnn*K more space.
$89	Could not parse File type or Aux File type.
$8A	No on-line volumes can be found.
$8B	Insufficient memory to perform desired update.
$8C	Boot Code file is the wrong size.
$8D	Bad ScriptFlag in script header.
$8E	Duplicate Folder Name.
$8F	Invalid Pathname Syntax *(special case)*

NOTE: Installer errors $84, $85, $86, $89, $8C, and $8D should never occur during execution of a properly written script. They are intended as an aid to script writers for debugging scripts.

Error $86 or $46 will give the first 32 characters of the source and destination pathnames so that script writers may identify which FileSpec field is at fault.

Error $87 indicates that required file matching (date-time and/or type-auxtype) failed, or that the needed file is missing.

Any other errors are merely reported back to the user by error number.